THE UNIVERSITY *of* EDINBURGH

PERFORMANCE MEASUREMENT AND EVALUATION OF TIME-SHARED OPERATING SYSTEMS

John Colin Adams

Ph.D.

University of Edinburgh

1977

CONTENTS

Acknowledgements

I would like to acknowledge the contribution made to my understanding of this subject by all the other workers in this field whom I have met and discussed this topic with over the past six years, unfortunately it would be impossible to name them all. However, I would like to give special thanks to Professor H. Whitfield for initially interesting me in this area and to Professor S. Michaelson for his continued encouragement throughout this work. I should also like to thank all the people involved with the EMAS project both in the Department of Computer Science and the Edinburgh Regional Computing Centre, especially Dr. G.J. Burns and Dr. A. McKendrick, of the ERCC, whose enlightened approach to system management allowed me to carry out the measurement experiments; G.E. Millard, B.A.C. Gilmore and all the ERCC staff involved in the definition of the interactive benchmark and the construction of the 'stimulator', and P.D. Stephens of the ERCC for all his help in topics concerning the EMAS resident supervisor. Finally I would like to thank my supervisor Dr. D.J. Rees without whose encouragement I should never have completed this thesis, and my wife Judy who not only put up with me, and my odd working hours, throughout this project without complaint, but also made such an excellent job of typing this thesis.

Abstract

Time-shared, virtual memory systems are very complex and changes in their performance may be caused by many factors - by variations in the workload as well as changes in system configuration. The evaluation of these systems can thus best be carried out by linking results obtained from a planned programme of measurements, taken on the system, to some model of it. Such a programme of measurements is best carried out under conditions in which all the parameters likely to affect the system's performance are reproducible, and under the control of the experimenter. In order that this be possible the workload used must be simulated and presented to the target system through some form of automatic workload driver.

A case study of such a methodology is presented in which the system (in this case the Edinburgh Multi-Access System) is monitored during a controlled experiment (designed and analysed using standard techniques in common use in many other branches of experimental science) and the results so obtained used to calibrate and validate a simple simulation model of the system. This model is then used in further investigation of the effect of certain

system parameters upon the system performance. The factors covered by this exercise include the effect of varying: main memory size, process loading algorithm and secondary memory characteristics.

CHAPTER 1

The class of computer systems addressed in this thesis is that of general purpose, time-shared, virtual memory systems. Within these, some form of operating system controls the sharing of a set of centralised computing resources - processors, memories, file storage devices - amongst a large community of users. Users interact with the system, and their programmes running therein, via keyboard like devices, rather than submitting their work on decks of cards, or rolls of paper tape, to some job reception desk whence they will receive their results sometime later (as in a batch form of operation). These systems also provide their users with some form of file system in which programmes and data may be stored, a large address space or virtual memory [Denning 1970] in which these programmes may be run and some mechanism whereby any user's programmes and data may be shared by, or protected from, other users. The range of work the users may carry out on such systems will not be restricted to any one particular language or class of operation as is the case in certain special purpose systems e.g. JOSS [Bryan 1967].

One of the major motivations for introducing such a form of system in the early 1960's

was a desire to make the use of computing more convenient to the programmer. The best way of achieving this would probably be to give each programmer his own processor with a very large main memory, however the cost of computer hardware at the time made this impossible. The solution adopted was to share a powerful mainframe with some form of virtual memory amongst several users, and to divide the available resources (CPU time, memory space, channel bandwidth) in such a way as to give each user the illusion that he had a whole, if less powerful, machine dedicated only to him. Subsequent studies [Gold 1969] have found that interactive use of computers is superior to batch use in problem solving, and with the current trend of dropping hardware costs relative to software costs this more efficient use of programmers' time will become more and more crucial.

The first time-sharing system, in which each ready to run programme is allocated a small quantum of CPU time in turn, was the Compatible Time Sharing System [Corbato et al. 1962, Crisman 1965] implemented at Massachusetts Institute of Technology on an IBM 7094. This was also the first true general purpose, multi-access system with users communicating with the machine via keyboard terminals attached by means of telegraph lines. A similar type of system -

the Cambridge Multiple-Access System [Wilkes 1973] was developed at Cambridge University on the TITAN computer. These two previous systems did not however provide virtual memory. The concept of virtual memory, in which the address space used by the programmer is split from that used by the hardware of the processor, also appeared in the early 1960's. This splitting of the address spaces allows each programmer to use an address space at least as large as, and often much larger than, the one available in the physical main memory of the machine. The two commonest mechanisms employed in providing virtual memory, either individually or together, are paging (introduced on the ATLAS computer [Kilburn et al. 1962] at Manchester University) and segmentation [Dennis 1965].

Systems which employ both time-sharing and virtual memory include: The Michigan Terminal System - MTS [Alexander 1972] produced at the University of Michigan on an IBM 360/67; the Multiplexed Information and Computing System - MULTICS [Corbato and Vyssotsky 1965, Glazer et al. 1965, Vyssotsky et al. 1965, Daley and Neumann 1965, Ossanna et al. 1965, Organick1972, Corbato et al. 1972] developed at M.I.T on a GEC 645; the TENEX system [Bobrow et al. 1972, Murphy 1972] implemented by BBN on a DEC PDP-10; CP/67 [Meyer and Seawright 1970,

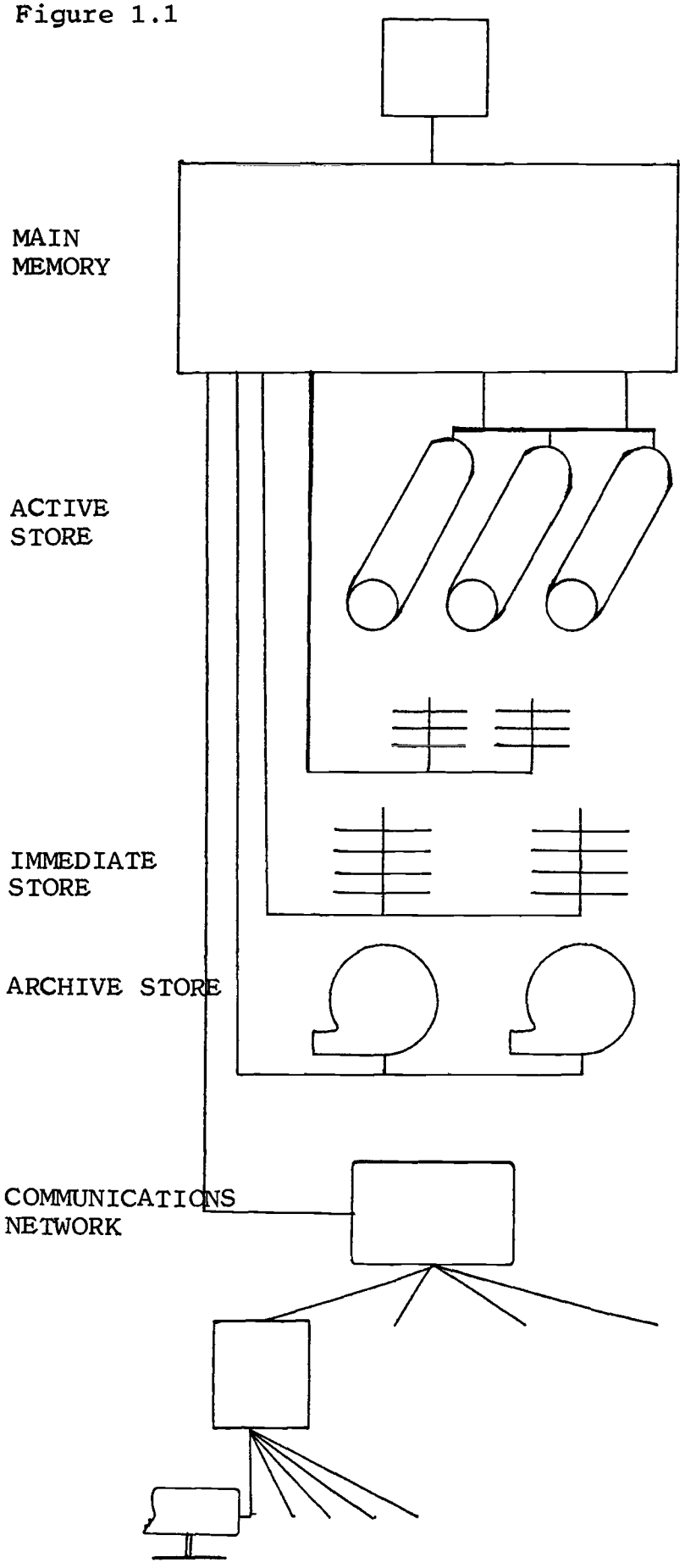
IBM 1970] produced for the IBM 360/67 and VM/370 [IBM 1972] produced for the IBM 370 series, both at the IBM Cambridge Scientific Centre.

The Edinburgh Multi-Access System

The system upon which most of the work reported in this thesis is based is the Edinburgh Multi-Access System - EMAS [Whitfield and Wight 1973, Rees 1975, Millard et al. 1975, Shelness et al. 1974, Wight 1975]. EMAS is amply described in the cited references, but as it plays such a central role in the succeeding work a brief description will be given here.

EMAS is a time-shared, virtual memory operating system implemented at Edinburgh University on an International Computers Ltd. System 4-75. The ICL System 4-75 is a byte addressed, third generation machine similar in structure and order code to the IBM 360/67. It offers virtual memory by means of segmentation and paging, the address space being split so as to present the programmer with 256 segments, each of up to 16 pages, each page of 4096 bytes. Figure 1.1 shows a typical EMAS hardware configuration at the time of the work reported.

Figure 1.1



TYPICAL EMAS HARDWARE CONFIGURATION

EMAS is written in the high level language IMP [Stephens 1974] and provides a virtual memory of 2^{24} bytes for a number of simultaneous processes (currently up to 63). The system maintains an on-line storage hierarchy of three levels, pages normally being held only at the outermost - immediate - level (currently formed by a 700 M-byte disc store) and are moved to the inner levels - active memory (currently formed from one or more two M-byte drum stores) and main store (currently formed by up to one M-byte of core storage) - as required. The user programme has no direct access to any Input/Output hardware, all management of the three tier storage hierarchy being carried out by the system and all unit record I/O being spooled. There is also an automatic archiving system [Wight 1975] which allows currently unused files to be removed from immediate store to archive storage (magnetic tape) and restored therefrom as required. A form of working set policy [Denning 1968] is used in the management of main memory. This is based on usage information obtained from read/write markers associated with each physical core page. Sharing is also supported at all levels of the on-line storage hierarchy.

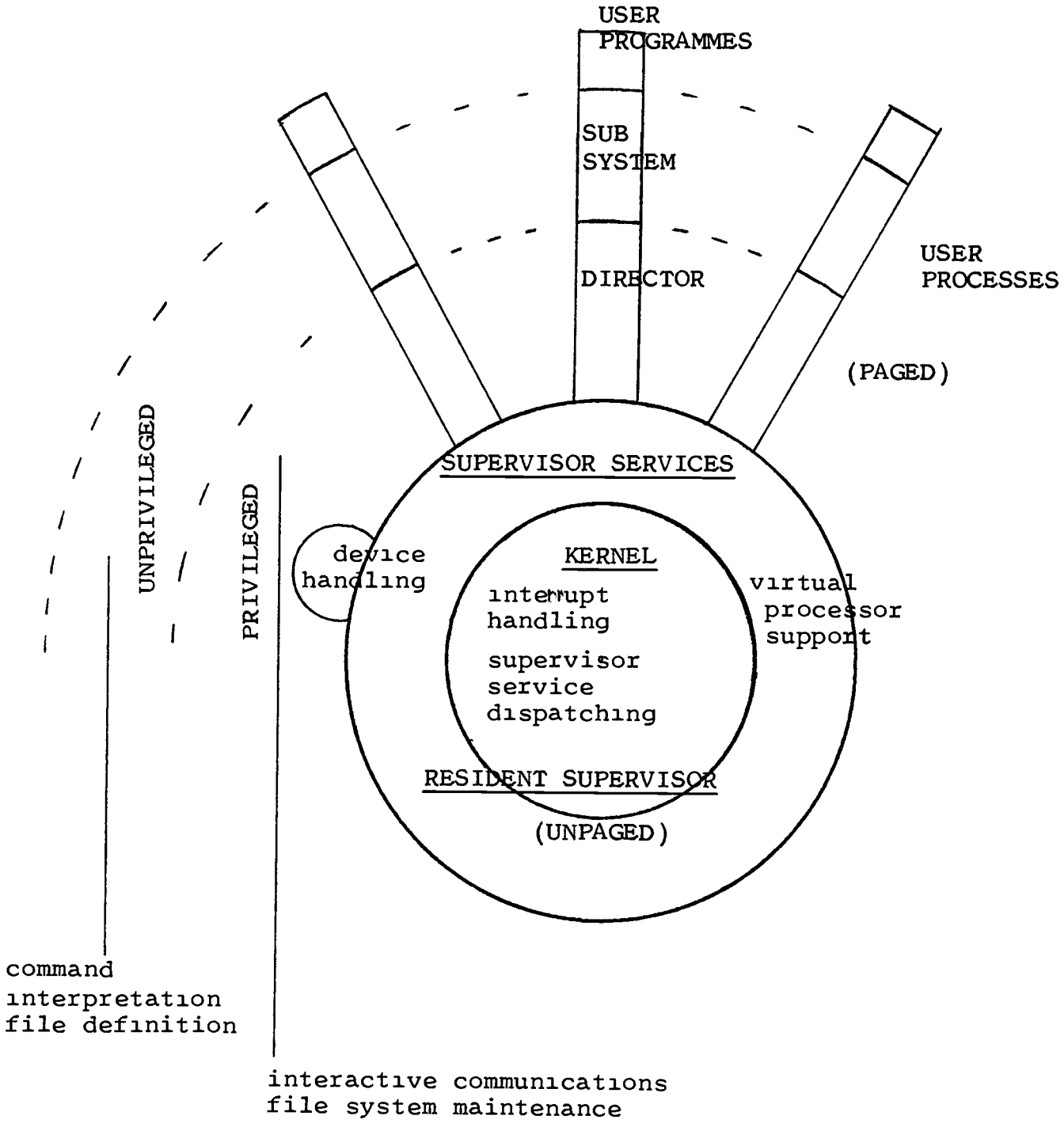
The operating system is itself hierarchically ordered and message based. The logical

structure of the system is shown in Figure 1 2. All communications between processes - both system and user processes - take place via a central message passing area All supervisor processes (or services) which have a message or request outstanding also have an entry in one central queue - the MAIN-Q One of the major functions of the innermost level of the system - the KERNEL is to remove entries from this queue and call the appropriate service. When there are no outstanding supervisor requests then the KERNEL will load the currently selected user process to the CPU where it will be allowed to remain for a maximum of a time-slice (100 milliseconds at present) at a time. The other major function of the KERNEL is to field interrupts and translate them into messages to the appropriate handling service.

One level out from the KERNEL are the supervisor services themselves. The services take care of two major functions:

Figure 1.2

EMAS LOGICAL STRUCTURE



DEVICE HANDLING - The handling of all paged I/O or interactive communications hardware attached to the system, scheduling transfer requests and carrying out all necessary device control.

VIRTUAL PROCESSOR SUPPORT - The allocation and management of the available resources (CPU time, main and active storage space and channel bandwidth) between competing processes and the management of process virtual memories.

The KERNEL and all supervisor services form the resident supervisor which is always in main memory and runs unpaged - using real addresses.

At the next level out run the user processes. Each user process consists of two levels: the paged supervisor, or DIRECTOR [Rees 1975], and the normal user process. DIRECTOR takes up 31 segments of the 255 segment virtual memory available to each user process (segment 0 is never used because of a

peculiarity of the hardware). It handles all interactive communications messages, maintains the file system and takes care of the allocation of immediate (tertiary) memory space. The resident supervisor knows nothing of files but merely handles page transfers. It is one of the functions of the DIRECTOR to associate virtual memory addresses with files resident in immediate memory when requested to do so by a user (the files are not permanently mapped into the virtual address space as is the case in MULTICS). Most of the DIRECTOR code and data space (interactive communications buffers and file indices) is shared amongst all user processes. The only unshared segment is the master segment which contains all local variables and tables for that process, in particular, one page of this segment - the master page - holds various tables and variables used by the resident supervisor and must always be in main memory when the process is on the CPU. Those segments which constitute DIRECTOR may not be accessed by normal user programmes (though the DIRECTOR may access the full virtual memory space) and those entries in the process' segment table are masked out when normal user programmes are running.

Running within the user level of the process is the subsystem [Millard et al. 1975] which

takes care of user command interpretation, file definition, linking and loading and logical I/O mapping. At a level out from this run the users' programmes. All commands on the system are merely external routines which have an entry in one of the user's libraries. A user may add new commands by compiling new external routines and making an entry in an appropriate library, or may call existing commands as routines from within his programme.

Certain 'executive processes' run at the level of user processes and perform such functions as I/O spooling, batch scheduling, archive storage control and the running of engineering test programmes. Though these are essentially user processes they have certain privileges and are scheduled slightly differently by the supervisor.

Scheduling Within Resident Supervisor

The majority of the work presented later will concentrate upon the workings of the resident supervisor and the scheduling algorithms implemented therein [Shelness et al. 1974]. An overview of these algorithms is now given.

All process scheduling within the system

is table driven from an entity known as the category table. Each process known to the system has assigned to it a category dependent upon the recent past history of that process. Associated with each category are the following attributes:

1) A set of resource constraints governing the amount of CPU time, main memory and active memory which each process of that category may consume during a period of main memory residency.

2) A priority level.

3) A time interval (known as the strobe time) associated with calculation of the working set.

4) A set of transitions to other categories dependent upon the actions of the process during its next main memory residency.

During the period covered EMAS had 20 different categories. The values contained in this category table are shown in Table 1.1. All normal user processes start in category 1 and thereafter use categories 5-20. Categories 2-4 are reserved for the

Table 1.1

EMAS Category Table (EMAS Version 802)

CAT	PRIORITY	CORE ALLOWANCE (PAGES)	A.S. MAX (PAGES)	A.S. MIN (PAGES)	NCY1	NCY2	NCY3	NCY4	RESIDENCY CPU (SECS)	STROBE INTERVAL (SECS)
1	1	50	80	50	17	15	11	14	1	0·125
2	1	20	80	50	3	2	2	2	0·5	0·5
3	1	30	80	50	4	3	2	3	1	1
4	1	50	80	50	4	4	3	4	2	0·5
5	1	20	80	50	8	6	5	5	0·5	0·5
6	4	20	80	50	10	7	6	5	4	1
7	4	20	80	40	10	7	8	5	10	1
8	1	30	80	50	11	9	5	8	1	0·5
9	4	30	80	50	13	10	6	8	10	1
10	4	30	80	45	13	10	7	8	6	1
11	2	40	80	50	14	12	8	11	1	1
12	4	40	80	50	16	13	9	11	10	1

(CONTINUED)

CAT	PRIORITY	CORE ALLOWANCE (PAGES)	A.S. MAX (PAGES)	A.S. MIN (PAGES)	NCY1	NCY2	NCY3	NCY4	RESIDENCY CPU (SECS)	STROBE INTERVAL (SECS)
13	5	40	80	50	16	13	10	11	12	1
14	2	50	80	50	17	15	11	14	1	1
15	4	50	80	50	19	16	12	14	10	1
16	5	50	80	50	19	16	13	14	10	1
17	3	60	128	64	20	18	14	17	2	0·5
18	4	60	128	64	20	19	15	17	7	0·5
19	5	60	128	64	20	19	16	17	5	1
20	3	62	128	64	20	18	15	17	2	0·25

A.S. MAX Maximum Active Store Allowance

A.S. MIN Minimum Active Store Allowance

NCY1 Next Category if Process runs out of main memory

NCY2 Next Category if Process exceeds CPU time allowance

NCY3 As NCY2 but has used less than the next smallest main memory allowance

NCY4 Next Category if Process goes to sleep

executive processes.

Figure 1.3 shows the major states and supervisor queues involved in the handling of processes on the system. Each process known to the system exists in one of three states:

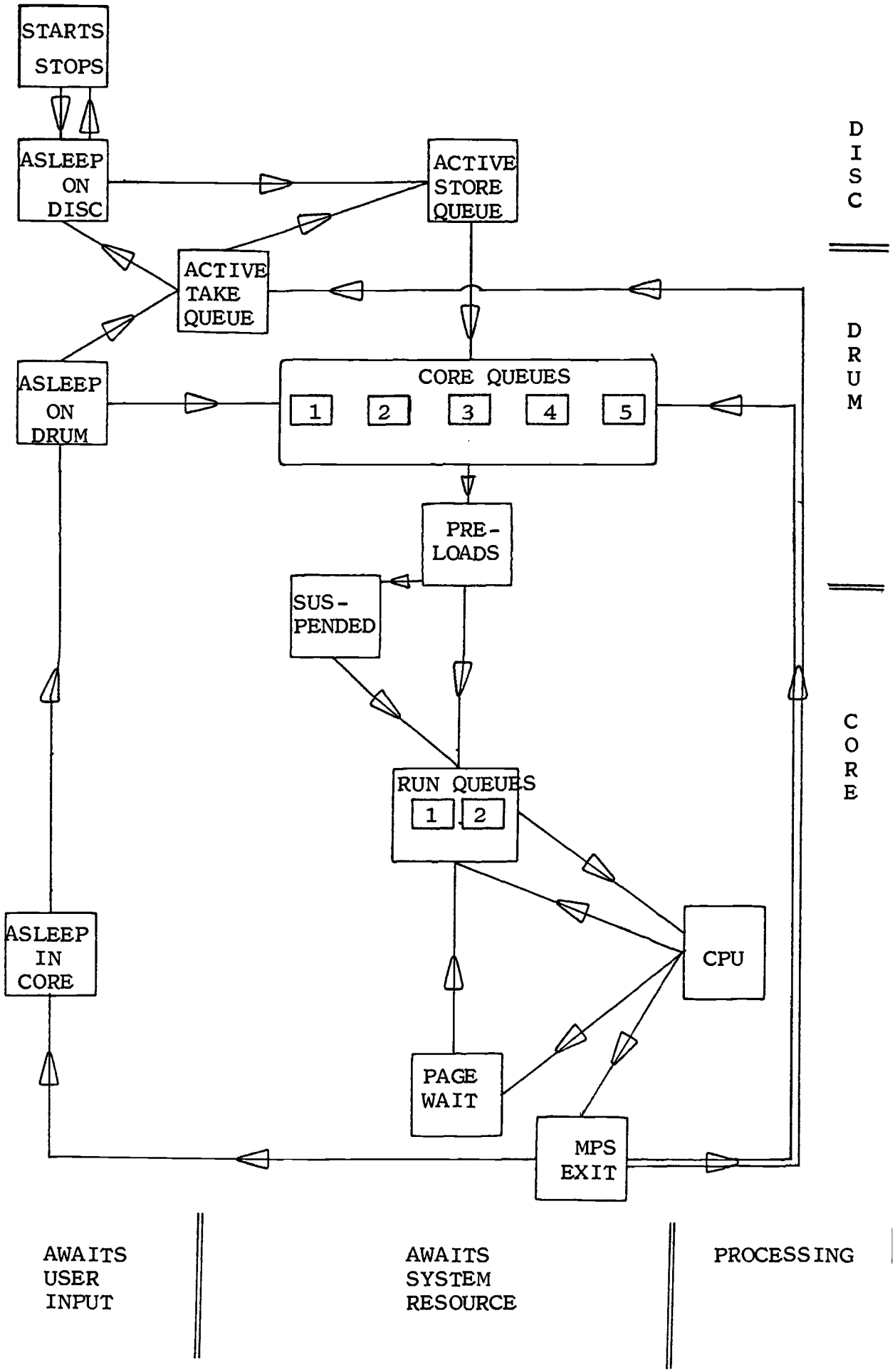
a) ASLEEP - awaiting user input or the freeing of output buffer space i.e. in terminal wait

b) AWAKE - awaiting allocation of some system resource

c) PROCESSING - on CPU

Each process will also be resident at up to a certain level in the storage hierarchy: immediate, active or central memory. Thus a process which wakes up resident only in immediate memory is first queued in the Active Store Queue to await an allocation of active storage. When an allocation of active store has been given (which at this point involves no identification of the particular physical active store pages to be used) the process will be placed in one of five core queues according to its current category's priority. These core queues are currently serviced

Figure 1.3 EMAS PROCESS MANAGEMENT MODEL



according to a priority scheme which assigns the probability of being selected as 39/64, 17/64, 5/64, 1/32, 1/64 respectively to the five priority levels. Once selected from its core queue the process is then held until it can be given its full allocation of main memory (again defined by its current category). Only when its full allocation of main memory is available may the process enter the multiprogramming set, and the contents of its current working set (which will always consist of at least the master page) will then be transferred (preloaded) into main memory. The system thus carries out a scheme of working set replacement. When all of the process' working set is resident in main memory (and not before) the process is placed on one of two run queues to compete for allocation of the CPU. All processes belonging to categories in the lower three priority levels go onto run queue one, whilst all processes in categories of the top two priority levels are placed on run queue two. The run queues are serviced according to an absolute priority scheme in which run queue one is always serviced first, and if any process from run queue two is holding the CPU when a process arrives for run queue one, then the arriving run queue one process will preempt the CPU process, even if that process has not completed a time-slice. Only processes which are ready to take

the CPU are held in the run queues. Once on the CPU the process may page fault and add a page to its working set from either immediate store, active store or main memory (for shared pages or new pages 'created' in main memory). The process may only hold the CPU for a maximum of a time-slice at a time.

Whenever a process has consumed a full strobe interval of CPU time during any residency, then its working set is recalculated and those pages no longer used are released. A process will remain resident in main memory until it goes to sleep or overruns one of its category allowances. It will then be rescheduled (perhaps into a new category) have its working set recalculated and be removed from main memory before being placed on an appropriate scheduler queue if it is still awake. Whenever a process is to be removed from active to immediate store it is first queued in the active take queue which essentially allocates channel capacity amongst those processes wishing to take this route (which involves pages being transferred first to main memory from active memory, then from main memory to immediate memory). A form of working set algorithm is also applied to the management of a process active storage allocation, the algorithm currently selects pages dependent upon usage over the last four main memory

residencies. There are four algorithms which may be used, the choice being dependent upon the current level of loading on the active store.

There are certain additions and modifications to the basic scheme. Any process which remains asleep for a long period of time (eight minutes) is removed from active store. Any process which remains awake for a certain interval (two minutes of real time) without interacting with the console is deemed no longer to be an interactive process and is placed in the penalty box. This means that when it comes to the front of a non empty core queue it will be returned to the rear of that queue several times (currently eight) before being removed. Whenever the process interacts with the user its penalty box status is removed. As it is extremely improbable that all the members of the multiprogramming set will be using their full main memory allowance at any given instant, and to take account of sharing, the main memory is over allocated by a certain amount. Another modification concerns preloading. If it is found that the next candidate for entry to the multiprogramming set cannot be given its full main memory allowance (even with the over allocation scheme) but that there is adequate physical space to allow that process working set to be

preloaded (and still leave some free space for use by other members of the MPS) this 'partial' preload is allowed to proceed. If the partial preload has completed and pages still have not been released to make up the process' full allocation, but its current allocation is greater than its working set size and a reasonable number of physical core pages are still free, then this process is allowed to enter the run queues, and acts as a normal MPS process which has a small main memory allocation. However, if this process' allocation is only equal to its working set size, or the number of physical core pages free is less than a safety limit, then it is suspended until adequate pages are released to give it its full allocation. There is, of course, a maximum of one partially preloaded process in the MPS at any instant, and this process has priority for the allocation of any freed central memory space. If a process overruns its current main memory allowance without ever having been strobed (i.e. having its working set recalculated) and, more than half its current pages were brought in by preloading, then there is a chance that the wrong pages were preloaded. To overcome this an EXTRA-STROBE (working set recalculation) is carried out at this point and if sufficient pages are removed, this process is allowed to continue.

To allow the available active storage space to be used fully, one of the replaceable disc drives is used as though it were a drum i.e. its storage capacity space forms part of the active storage. Allocation of active storage pages is handled so that all the drums are considered as though they formed a linear array of pages with this replaceable disc (known as the pseudo-drum) forming the higher addresses. The lowest free page available is always allocated first, thus the drums which correspond to the lowest active store addresses are kept as fully used as possible.

The main memory management scheme only allows a process to enter the MPS if it is estimated that there is sufficient central memory available for it to run efficiently. All decisions on the management of that process' allocation are then reached with consideration being taken only of that process' behaviour, and any process which is found to have a working set larger than its current main memory allocation is removed. This completely removes the phenomenon of thrashing [Denning 1968] which is due to an overcommitment of main memory. It also provides for the time-sharing of main memory by placing limits on the amount of time any process may remain in main memory. The algorithms are designed to favour highly

interactive processes by time-sharing the main memory and by the priority scheme which gives more residency periods to processes which require smaller amounts of main memory and very little CPU time.

Quantitative Evaluation Techniques

Performance evaluation is generally carried out for three major reasons [Lucas 1971]:

1) The selection of a new system - choosing from a set of possible alternatives which system best meets a user's performance/cost specifications.

2) The projection of the performance of a new system - estimating the performance of an as yet un-implemented system i.e. as an aid in the system design process.

3) The forcasting of the impact of possible changes in an existing system - changing a hardware or software component or the user load applied to the system i.e. system tuning or balancing.

Quantitative evaluation has grown

increasingly more difficult with the evolution of time-shared, virtual memory systems. The systems themselves have grown more complex and the range of programmes executed upon them has become wider and more varied.

In the earliest days of computing a simple figure of merit was considered adequate as a means of judging the performance of any system. In the case of 'scientific systems' the figure of merit would often be based on the raw power of the central processing unit. This number could be obtained by calculating the execution time of a certain instruction stream, the mix of various classes of instruction included would represent a rough characterisation of the anticipated workload, or be drawn from some generally accepted mix [Gibson 1970]. Meanwhile for more 'commercially orientated' data processing systems the figure of merit would be based upon some measure of I/O throughput capacity.

As early operating systems were introduced their batch type of operation was often judged in terms of the time taken to process a chosen collection of jobs or benchmark. The benchmark would again form a characterisation of the expected workloads in terms of the proportions of the types of jobs it

contained. Other simple one figure measures such as job throughput rate or processor utilisation level were also often used.

However, as the architecture of the systems has become increasingly more complex it has become clear that no single figure of merit, or even any small number of figures of merit, will be adequate to describe a system's performance [Grenader and Tsao 1972], though several continue to be proposed [Merill 1975, Steven 1975].

Within an interactive system the only pure performance metric which every user applies is that of response time. Response time is loosely defined as the time a user has to wait, from the moment he gives a command to the system, until the moment at which he receives an answer. The distribution of these responses will be of interest rather than simply the mean or median response. Studies [Miller 1968] of human reactions in the man-computer interaction cycle have shown that if a response is greater than two seconds then the user begins to lose concentration, and if a response is greater than 15 seconds then the use of the computer ceases to be interactive.

No performance measure upon such systems is meaningful unless accompanied by some measures of the outstanding load upon the system e.g. processor utilisations, memory utilisations, number of simultaneous users, mean working set sizes, mean time between page faults, supervisor overheads. The problem of evaluation is not just to attach some figure of merit to a system or particular system configuration, but to attribute the observed performance to the various contributing factors and identify those factors which are most significant. Performance and load measures will vary from system to system and will depend upon the problem being addressed. Suitable metrics for time-shared virtual memory systems will be introduced later.

The two major aids to evaluation are modelling and measurement.

Modelling

Because of the inherent complexity of the systems under consideration the technique of modelling which produces a much simplified, abstract representation of the system has an obvious appeal. Indeed, no evaluation of a system could proceed without at least the existence of some conceptual

model of how the system functions. Figure 1.3 could be regarded as such a conceptual model of the working of the EMAS process scheduling scheme. The value of a model may not only lie in the quantitative results it produces, but the actual formulation of the model itself, involving as it necessarily does the stripping away of a mass of detail, may reveal the major components of the system and their interrelationships.

Quantitative modelling techniques fall under two headings:

Simulation models

Mathematical models.

Simulation models [Leroudier and Parent 1976] consist of computer programmes, often written in a special purpose simulation language [Dahl and Nygaard 1966], or using a simulation package written in a high level language [Dimsdale and Markowitz 1964]. The representation of the system being modelled is embedded in the simulation programme. Using this technique it is possible to model all the major mechanisms involved in computer systems e.g. parallelism, variance in user programme characteristics, storage capacities, various servicing disciplines and various service time

characteristics. However, simulation is often criticised for being expensive and time consuming in both development and run times. The time and expense involved in certain cases may, in fact, make this approach impractical. However when this is not the case simulation does provide the ability to model whatever phenomenon may be considered significant.

Mathematical modelling mainly centres round probabilistic models and more particularly queueing theory. There has been considerable work in this area. The research has evolved from the study of single queues [McKinney 1968, Chang 1970] to the study of various networks of queues [Jackson 1963, Gordon and Newell 1967, Buzen 1973, Gelenbe 1975, Baskett et al. 1975, Gelenbe 1976]. Following from the classic analysis of CTSS [Scherr 1965] there have been attempts to apply such models to the evaluation of time-shared, virtual memory systems viz MTS [Moore 1971] and MULTICS [Sekino 1972] but it is only recently that such models have been put to practical use with the development and extensive use of a model of IBM's VM/370 system [Bard 1975, Bard 1976, Bard 1977]. Queueing network models still suffer from several limitations: there is no direct way to model storage, service disciplines

and service time distributions are still limited. However, they may provide a useful means of studying the gross performance characteristics of such systems.

Measurement

The other major aid to evaluation is that of measurement and experimentation on existing systems i.e. the empirical approach. The only way in which significant system phenomena may be identified in the first instant is through a procedure of empirical evaluation. Measurements from such a process may then be used in the essential step of validating current models and suggesting changes in future models of the system.

Several drawbacks to such an approach do exist. It is often difficult to obtain accurate measurements of particular phenomena of interest due to inadequate system instrumentation, or due to gross interference caused by the measurement technique. The opposite extreme is also often a problem - the sheer mass of data produced by some measurement tools masking the trends the experimenter is searching for. Measurements taken upon an operational system will depend crucially upon the characteristics of the workload existing at the time

the measurements were taken. These changes in user workload which take place from day to day, or hour to hour and minute to minute, often prevent the acquisition of a consistent set of measurements, from which changes in performance may be attributed to specific system changes. A rigorous approach to system measurement is one of the necessary paths to be followed when attempting to discover just how such systems do function.

The ideal approach to evaluation is an iterative one with results from a controlled set of experiments being used in a model which, when validated and calibrated by this data, will suggest new areas for experimentation.

The main aim of the work carried out in this thesis is to increase, in some way, the understanding of the mechanisms at work in time-shared, virtual memory systems, and to be able to quantify the impact of any major component upon the overall system performance. This is carried out by the evaluation of the structure (and design) of one particular system (EMAS). The evaluation is thus empirically based and concentrates upon the techniques and aids necessary in such an exercise. The monitoring tools required are discussed first

(Chapter 2). Then the elements necessary to carry out a programme of controlled experimentation on such systems are described and the execution of such an experiment is reported (Chapter 3). The results from this experiment are presented in detail (Chapters 4 and 5) and used in the calibration and validation of a simple simulation model. This model is then used in the further investigation of certain of the parameters affecting system performance (Chapter 6). Considering the three main areas of application of performance evaluation given at the beginning of this section, the techniques used and the approach taken fall under heading 3 - system tuning and balancing. However the results obtained and the techniques applied will also be of use in the other two areas.

Chapter 2

A comprehensive and effective set of monitoring tools is an essential aid in any empirical investigation of a system's performance. An ideal monitoring aid would be flexible and have the ability to obtain all required data (and only that data required) with absolute accuracy. This ideal monitor would not, of course, interfere with the system in any way either by adding to the supervisor overhead, or changing the behaviour of user processes. Unfortunately, in the case of time-shared, virtual memory (T.S.V.M.) systems such a monitor does not exist. In this chapter available monitoring techniques are reviewed, and those implemented in the EMAS resident supervisor are described.

Before any measure is carried out a clear view must exist of exactly what data is required and what use this data is to be put. Any possible interference caused by the method of measurement must also be known and taken into account. Data may be obtained upon the performance of the system itself or the behaviour of the user processes running thereon, i.e. the workload. The distinction between workload and pure performance measures is often blurred, and

the two are always related. Typical system performance results are often presented on:-

a) Response time distributions.

b) Utilisation levels of major system components (e.g. CPU's, memories, channels, supervisor modules).

c) Distributions of queue lengths or wait times for various system resources.

Typical measures of user process characteristics include:-

a) Distribution of the time the processes spend in terminal wait state (e.g. think times).

b) Space requirements of processes at various levels of storage hierarchy.

c) Patterns of access within virtual memories (e.g. size distribution and contents of working sets, distribution of times between page faults).

d) Distribution of resources required by each interaction.

e) Distribution of interaction classes.

In the following, the terms "target system" or "host system" will be used to mean the system being measured or experimented with.

Monitoring Techniques

A comprehensive review of current monitoring techniques exists in the literature [Nutt 1975], so only a brief summary of the advantages and disadvantages found in the major classes of monitor is given here.

Three classes of monitor exist:-

1) Hardware Monitors

2) Software Monitors

3) Hybrid Monitors

1) Hardware Monitors

A Hardware Monitor consists of a distinct electrical device (generally with its own clock and storage media) connected to the target system's hardware by a set of one or more probes. Signals received via these probes are interpreted by the device and data is then analysed on line, or logged (usually to magnetic tape) for later off line analysis. The probes used are usually of such a design that they cause no significant perturbations in the circuitry to which they are attached. This gives the hardware monitor its great advantage over all other techniques: it is essentially non-interfering, inducing no supervisor overhead or change of user behaviour in the target system. The accuracy obtained by this method is also usually dependent upon the precision of clock incorporated in the monitor, and not upon the clock facilities in the host mainframe.

The complexity of such devices varies greatly from the extremely simple - monitoring the existence of a single signal (e.g. a trace chart recorder connected to a processor's idle light [Stang 1969]) - to the other end of the spectrum where a fully interactive mini computer is

employed - with special computational as well as interface hardware, capable of simultaneously recording and analysing a very large number of interrelated events [Aschenbrenner et al. 1971].

Such monitors have been found very useful in obtaining summary data such as utilisation levels and degree of overlap on certain hardware components (e.g. CPU's and channels) or execution counts on the instructions in the mainframe's repertoire [Schreiber 1976]. However, it is often impossible to establish relationships between the data obtained and the causes for such levels of performance - user behaviour patterns and software scheduling algorithms. On more complex mainframes the correct placement of probes will become more difficult, and skilled engineering guidance will be required. The mainframe will also probably have to be taken out of service for a time whilst such a device is attached. With the introduction of mainframes using more and more Large Scale Integration (i.e. machines such as the Amdahl 470/V6) the placement of probes will become more and more difficult, and certain data may no longer be available for collection by this method.

The characteristics of hardware

monitors would seem to make them best suited as an aid where the pure performance of the hardware only is of interest e.g. counts of different types of instructions, degree of overlap of certain hardware devices. In the case of T.S.V.M. systems, where the complex characteristics of the user workload must always be taken into account, the use of pure hardware monitors alone is of limited value. They have, however, been applied to some time-shared systems such as CDC's Kronos system [Lindsay 1976]. The advantages of hardware monitors seem better suited to special purpose systems where a regular, well understood workload exists [Partridge and Card 1976] in such an environment they may even be used as an aid in programme optimisation [Fryer 1973]. Several types of hardware monitor are now commercially available.

2) Software Monitors

Software monitors provide an extremely flexible and popular method of obtaining performance data. They will, however, always have the great drawback that they necessarily interfere with the target system. They form part of the system, occupy memory space for code and data, consume processor power in execution and often use channel capacity in storing

data. The accuracy of any software monitor will usually be limited by the resolution of the hardware clock available on the mainframe.

A great range of software monitors have been implemented on various systems (indeed nearly every system contains a software monitor in terms of the accounting log). As a broad classification they can be divided according to their recording discipline into:

a) Sampling Monitors

and

b) Continuous-recording or event monitors

and, according to their storage discipline (i.e. the way in which data is disposed of once it has been collected), into:

c) Accumulating monitors

and

d) Tracing monitors.

a) Sampling Monitors

Sampling monitors are perhaps amongst the simplest to implement, and should impose the least

overhead on the target system. As the name indicates, the monitor is only activated at certain times, either at regular intervals using some form of alarm clock interupt, or by the occurrence of some system event, such as the idle process gaining the CPU. The metering routine thus activated will then obtain the required data and save it. This routine is normally distinct from the rest of the target system and so has the advantages of modularity (easy removal or modification). Also, as they are not active all the time they should impose less of an overhead than other monitors. The argument against using a sampling technique is that the accuracy will depend upon the number of samples and the randomness of the sample. Very few sampling monitors obtain their samples at truly system independent random intervals, so the result could be affected by periodic or other phenomena within the target system. This could have a very significant effect upon the accuracy of the results obtained. These monitors have been used in the investigation of code utilisation by sampling the programme counter [Waite 1973], and are often used to obtain approximate distributions of system queue lengths [Jalics 1973, Gonzales 1975].

b) Event Monitors

Event monitors are usually formed by a set of software probes scattered throughout the operating system and activated for periods of time by the setting of a group of trigger variables. These probes are necessarily scattered throughout the system, and thus not easily modified. Although data is only gathered when a trigger is set and the flow of control passes a probe, the trigger must be tested every time the probe is encountered, which means there will be a certain overhead even when no data is being collected. Event monitors, however, do not suffer from any suspicions about sampling accuracy, their accuracy only being limited by the resolution of the clock and the speed of the probe.

Storage Discipline

Monitors may be further classified according to their actions on obtaining a particular item of data. They may integrate this item into a table in main memory holding a summary of the performance data (accumulating monitors). This involves carrying out a small amount of processing on each item of data when it is collected. The accumulated table is then output (perhaps involving further

processing) regularly after comparatively long periods or on demand. The alternative is to do no on line processing on collection of data, but to output each item immediately, usually with some form of time stamp (tracing monitors). The accumulating method will tend to use more CPU time and code space - though a tracing monitor will use CPU in organising buffers and transfers. The table space used by an accumulating monitor tends to be a constant overhead, whilst tracing monitors may claim buffers from a system wide pool only for the duration of the measurements. Tracing monitors will consume channel capacity, often require exclusive use of a device (e.g. tape drive) and frequently produce great volumes of output. However, the data so produced allows greater flexibility as it may be analysed in several different ways to produce a variety of results.

Software monitoring is certainly the most popular method of measurement. It involves no acquisition of additional hardware, and can usually be implemented easily by the system programmers. They also have the advantage of being able to observe the cause and effect of certain transient events which a hardware monitor cannot. Software monitors are normally highly system dependent, though the principles involved may be transportable between different

operating systems, the monitor itself rarely can.

3) Hybrid Monitors

The logical merging of both hardware and software monitoring techniques results in the most recent monitoring method - that of the hybrid monitor. In this method a complex hardware monitor, usually consisting of a mini computer with associated probes, is however also attached to the host system as a normal device via some form of channel [Rudd 1972, Aschenbrenner et al. 1971, Estrin et al. 1972, Schwemm 1972]. This allows software monitoring aids implemented within the system to communicate with the mini computer. Thus whilst the majority of the data may be obtained in a non interfering fashion by the hardware monitor part, further information, allowing this data to be associated with various phenomena within the system, may be produced by the software aids communicating via the channel. This method does, of course, suffer from drawbacks of both hardware and software monitors: engineering knowledge is required for the correct placement of the probes; the host system may have to be taken out of service for the attachment of such a device; the software aids will necessarily interfere with the system; much knowledge of the software structure will be necessary for the

gathering of the correct data in the most efficient fashion. However, hybrid monitoring should still reduce overhead, and with many of the large mainframes now being produced, such as the DEC KL 10 and KL20 systems [DEC 1977], containing mini computers with access to most of the important registers and parts of the memory (i.e. a possible built-in hybrid monitor), it would seem to indicate that greater use could be made of hybrid monitors in the future.

One class of system performance measurement devices not covered here is that of the remote terminal emulator. This will be considered in the next chapter.

Virtual Memory System Monitors

The majority of instrumentation reported on these systems is carried out in software. Very little use appears to have been made of hardware monitors, almost certainly because of this difficulty in establishing relationships between observed performance and the factors which contribute to it. One reported case of what may be classified as hybrid monitoring does take place on MULTICS [Saltzer and Gintell 1970] with a PDP-8 being used with special access to the host systems tables and some registers.

However, as the data rate between the monitor and the host system is very low (less than 60 words/second) the full potential of this technique has probably not been realised.

As the behaviour of user processes is of such interest an ideal monitor would be one which allows the collection of data on process behaviour as well as the manner in which processes are handled by the scheduling algorithms. An event trace monitor which records an event each time a process moves significantly either within its virtual memory or within the system queues would appear to be one solution. The Data Collection Facility [Alexander 1975, Pinkerton 1969] on MTS is such a monitor. Implemented within the code of the resident supervisor the DCF allows the tracing of a set of events of one or more specified processes. The type of events which may be recorded allow data to be obtained on:

i) The queueing and removal from queues of processes by the supervisor.

ii) The changing of status of monitored processes.

iii) All aspects of page movement in and out of physical core, and the migration of pages to the outer levels of the hierarchy.

iv) The claiming and freeing of pages in virtual memory.

v) All interrupts generated on the system.

vi) The opening of files by processes.

vii) The starting and stopping of user tasks on the system.

A very comprehensive set of possible data items. As MTS is written in machine code some difficulty is involved in adding new events [Alexander 1977]. The vast amounts of data collected during any run are recorded on magnetic tape for off-line analysis. A data reduction programme - the Data Analysis Programme - is also available to aid the investigator in the interpretation of the data. A very sophisticated set of monitoring aids have been built into IBM's VM/370 system [Callaway 1975], allowing both sampling and event trace monitoring at various levels of detail in the system. This also has an associated Statistics Generating Programme to aid analysis.

The VM/370 performance monitor may be bought by customers running VM/370 to assist in tuning and balancing of their system.

MULTICS contains a variety of monitoring facilities to aid in the measurement of process characteristics [Saltzer and Gintell 1970]. Surprisingly, however, no generalised event trace monitor has ever been implemented, although a comment is passed in the Saltzer and Gintell paper that one would have been useful. The monitoring aids which have been implemented include:

i) A sampling monitor accumulating distributions of the segments used.

ii) A count which may be kept of all missing pages and segments encountered whilst executing a particular segment.

iii) A missing-page trace of the last 256 page faults produced by the monitored process (held in a ring buffer).

For the gathering of raw performance statistics on the system (i.e. utilisation levels or queue length distributions) MULTICS makes use of the Graphic

Display Monitor which is essentially a PDP-8 with access to certain of the host mainframe's registers and tables. This continuously displays all system queues and arrays, showing execution time profiles for supervisor modules. A count and total CPU time expended in certain supervisor modules is also accumulated.

A very sophisticated event monitor has been implemented on the TENEX system [Gonzales 1975] for the gathering of system performance data. This allows the definition of events to be monitored and the switching off and on of data collection to be carried out from a normal user process via a set of special supervisor calls and a password scheme. The probes which collect the data and the tables in which the data is initially accumulated are part of the resident supervisor, though the data may be transferred to the user process' file when desired. This contrasts with the considerably more rigid data storage regime of the MTS-DCF which, though obtaining a more general and more accurate range of data (1 millisecond clock in the TENEX scheme to a 13 microsecond clock on MTS), can only be controlled from the operators console, and always outputs to a specified magnetic tape drive. An event monitor which accumulates distributions of various queues and timings has been implemented on the

TOPS-10 system [Jalics 1973]. Both of these event accumulating monitors are used more for obtaining performance statistics on the system than on the behaviour of the user processes.

Monitoring Aids on EMAS

The purpose of the monitoring aids implemented in EMAS was to give performance data on the system which would be of use in investigations of the architecture and algorithms employed within the system, as well as being of use in tuning the system in practical use. No hardware or hybrid monitoring aids were available, and all monitoring has been carried out by software techniques. The clock used throughout was that provided on the ICL 4-75 mainframe with a precision of 6·5 microseconds. As EMAS was designed as an extensible system on which the user has the capability of writing his own subsystem or even file system, all the performance monitoring aids considered here were implemented within the innermost level, i.e. that of the resident supervisor. Various other monitoring aids have, of course, been implemented at other levels [Adams and Millard 1975]. The entire system is written in the high level language IMP. The advantages accrued from this fact cannot be over-emphasised. Apart from allowing for the easy implementation of

software probes, the modularity of the system allows great flexibility and ease of change, with only the module which has actually been changed needing to be recompiled. The compilation and linking of a modified system, taking in the order of fifteen minutes (real time), is extremely fast for a system of this complexity and size.

CPU Time Utilisation

A profile of CPU time utilisation was considered to be vital to such an investigation. The vector of CPU time spent in major states (SUPERVISOR, USER, IDLE) would, of course, be one of many important parameters to be considered. Furthermore, as the supervisor activity within this class of system is inherently higher than that in some other forms of systems, it would be of interest to know in which modules of supervisor code most of the CPU time was being spent.

The message based nature of communication between EMAS supervisor services lends itself well to the monitoring of these variables. A simple change in the kernel where requests are unstacked from the Main-Q allows a count to be kept of the number of calls made on each service, and the total CPU time expended

between calling the service and returning from it. As the services run uninterruptably this gives a very precise account of where, within the resident supervisor, time is being spent.

Whenever the supervisor finds that there is no user process in central memory in a ready to run state and no supervisor requests outstanding which can be fulfilled i.e. that the system is idle, then process $\neq\neq$ 0 - the idle process - is loaded onto the CPU, and executes an idle loop until some form of work arrives. This process is essentially handled as a normal user process, and has the CPU time it consumes recorded in its entry in the process list. Thus an accurate measure is obtained of the time the CPU is idle. A further split is made in the idle time between time in which no user processes are active (i.e. no user process is awake - true idle time) and time in which user processes are active, but for some reason none could proceed - blocked time. The CPU time not being used by the supervisor or the idle process within an interval is that consumed by user processes and unaccounted kernel time. In normal analysis this time which is the time spent translating interrupts to requests on appropriate service and on handling the MAIN-Q itself, is attributed to user processes. The time consumed

in this will be further discussed in Chapter 4.

The dumping of the data arrays involved takes place at systems close-down or on the setting of a system test flag from the operators console. First in raw form showing the total number of entries to each service, the total time spent in that service (in seconds) and the average time per call (in microseconds) [Table 2.1]. To minimise the insignificant entries nothing is printed on services which use less than one second during a session. The data is also processed on-line to obtain the CPU breakdown between major states (Supervisor, Idle, User) and the breakdown by function within the supervisor [Table 2.2]. A machine readable form of this data exists in the system main log should further processing be required.

Interference caused by this measurement consists of:

a) Two arrays of 256 bytes each to hold the data.

b) A small number of extra instructions in the KERNEL to gather the data.

Table 2.1

SAMPLE OF RAW CPU MONITOR DATA

METERING INFORMATION
IDLE TIME(SECS)= 56
NO WORK AVAILABLE TIME(SECS)= 0

SERVICE	COUNT	TIME	MUSECS
3	22	0	0
4	186	0	0
6	54264	133	2450
7	1905	2	1049
8	19145	30	1566
9	13556	12	885
10	16088	0	0
11	1658	0	0
14	262	0	0
26	131	0	0
27	131	0	0
28	1310	0	0
29	40145	179	4458
36	800	1	1250
38	5	0	0
39	2	0	0
40	3	0	0
41	14016	22	1569
42	79	0	0
50	75204	54	718
52	15	0	0
54	8544	3	351
55	9310	35	3759
57	63812	30	470
58	102497	79	770
59	49309	62	1257
63	6551	70	10685
64	47567	20	420
65	803	0	0
66	624	1	1602
67	2065	0	0
68	7	0	0
69	1048	1	954
70	14938	13	870
73	272	0	0
77	256	2	7812
78	8282	6	724
79	279	0	0
80	14400	17	1180
81	103	0	0
82	79	0	0
84	262	7	26717
85	249	0	0
86	3606	3	831
88	2	0	0
89	442	0	0
95	16	0	0
96	148	0	0
97	37	0	0
100	1	0	0
102	40	0	0
103	692	0	0
108	22	0	0
109	13567	7	515
110	9131	6	657
112	131	0	0
115	11	0	0
117	3729	2	536
119	9827	8	814
120	188	0	0
128	131	0	0

Table 2.2

SAMPLE OF PROCESSED CPU DATA

	TIME	% TOTAL TIME
TIME IN USER PROCESSES	1024	54.32
SUPERVISOR TIME CHARGED	740	39.25
SVC'S	128	6.79
PAGETURNS	612	32.46
UNCHARGED SUPERVISOR TIME	65	3.44
IDLE TIME	56	2.97
NO WORK	0	0.00
BLOCKED	56	2.97
TOTAL TIME	1885	100.00

ANALYSIS OF SUPERVISOR TIME

VIRTUAL MEMORY SUPPORT DRUM TRANSFERS (6,29)	312	16.55
DISC TRANSFERS (7,8,32-41)	55	2.91
CORE LOADING(55-6-8-9,63-4)	266	14.11
DRUM LOADING (73-80).	25	1.32
PROCESS CONTROL (70)	13	0.68
TIME SLICING (50)	54	2.86
FILE SYSTEM (54,85-6)	6	0.31
SVC PARAMETER PASSING (57)	30	1.59
COMMUNICATIONS (9,100-19)	35	1.85
POLLING DEVS (14,27-8,69,72)	1	0.05
MAGTAPES (5,15-23)	22	1.16
MISC.	0	0.00

c) Some code to print out the data, approximately 1500 bytes to produce the raw form, and a further 2000 bytes for on-line processing. There is no reason why the second routine should not be moved out of the resident supervisor and modified slightly to analyse the raw data from the main accounting log.

d) The CPU time consumed by this method will, of course, vary depending upon the level of supervisor activity, but has been measured to be less than ·5% of total time during normal use. The time consumed in dumping the accumulated data and the analysis is of the order of half of a second.

This aid gives very accurate data on time spent in supervisor services and idle state during any interval.

Event Trace Monitor

Clear and accurate data was required on the characteristics of running processes and the manipulation of these processes by the system. The best way of obtaining such information is an event trace monitor along the lines of the MTS - DCF. The design

of such a facility is very straightforward. A set of probes is incorporated in the resident supervisor software. These probes are activated by the setting of a system test flag and, when triggered, call a data gathering routine which adds a time stamp and transfers the data to a buffer. The data gathering routine also organises the transfer of filled buffers to the backing store used.

The placement of probes within the software was relatively easy, aided by the modular design of the system and the high level nature of the language it was implemented in. A tracing scheme had been incorporated during early system development to aid system debugging, and several of the significant events overlapped. The probes are implemented as a set of calls on the data gethering routine, conditional upon the setting of a particular flag. The parameters of these calls contain the relevant data. Originally it was planned to use magnetic tape as the storage media, but it was discovered that the replaceable disc unit, used in normal operation as the pseudo-drum, in fact only used the first hundred cylinders on that pack (the space available on a normal drum), leaving eight hundred pages of storage space free. The event trace monitor thus stores its data in this fixed area, though it would be a simple change to make it dump the data

elsewhere (e.g. to magnetic tape).

The operational procedure involved in using the event trace monitor is as follows:

1) The monitor is switched on from the operator's console by setting a system test flag to a mask value showing those events which are to be monitored. If monitoring is required on only one process, then the test flag must be set to the process list index for that process (after a prompt has been sent).

2) The monitor then claims some buffer space (currently two pages), and activates the probes.

3) When the monitor has filled its data area (currently 800 pages) or the system test flag is reset, the monitor is switched off, the probes de-activated, and the buffer space returned to the system. The number of pages of data accumulated and the number of gaps in the data (caused by not having a buffer ready) are printed in the main log.

4) The monitored data may then be transferred to a normal EMAS file for analysis by making use of a

utility programme run in one of the privileged EXECUTIVE processes. As the EMAS resident supervisor has no knowledge of files, but only manipulates pages in process virtual memories between various levels in the storage hierarchy, it would have added unnecessary complexity and overhead to have data transferred directly from the monitor to a normal EMAS file within a user process. It would also have interfered with the operational characteristics of that process whose file was being used thus.

The data recorded whenever a probe is triggered always has the following format:

Word 1 - Consists of four byte fields:

i) The event identifier.

ii) The length in words of this data record.

iii) The process to which this event pertains.

iv) The process holding the CPU.

Word 2 - Consists of the current value of the 4/75 clock register.

Words 3 + - Hold the data parameters for this event up to 253 data words per record.

The model of the system used when deciding which events were significant and should be monitored was the Process Management Model. Where the monitorable events in the standard version of the monitor correspond to movements on the P.M.M. graph, the event identifier is shown circled in Figure 2.1. A list of events which may be monitored in the standard version of the monitor is shown in Table 2.3. These events fall under the following broad headings:

a) Paging Events

These events enable the collection of data on the virtual addresses used by the process, the distribution of working set sizes etc. as well as the distribution of wait times caused by the various types of paging going on within the system. Events may be recorded whenever:

- A process is elected to the Multiprogramming Set and begins a preload - the master page at least is always preloaded (event ≠≠ 3). The number of pages preloaded and the number of transfers required is recorded.

Figure 2.1 EMAS PROCESS MANAGEMENT MODEL SHOWING TRACE MONITOR EVENTS

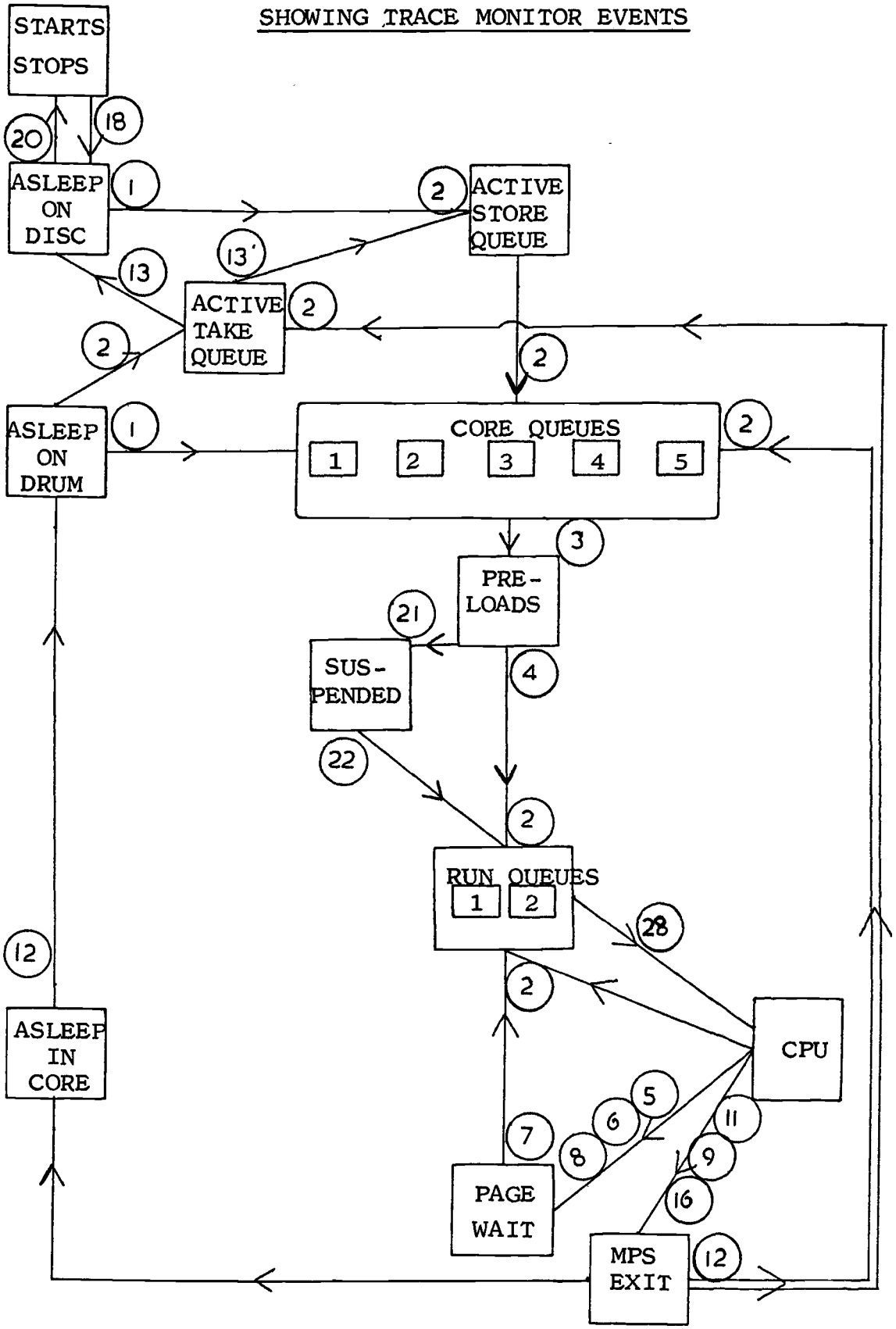


Table 2.3

STANDARD EVENT TRACE MONITOR - LIST OF EVENTS

IDENTIFIER	EVENT
1	Process wakes up.
2	Process put onto scheduler queue.
3	Process enters Multiprogramming Set.
4	Process completes preload.
5	Process page faults - page on tertiary memory.
6	Process page faults - page in secondary memory.
7	Page faulted page arrives in main memory.
8	Process page faults - page in main memory.
9	Process overruns a category resource limit.
10	Process completes strobe interval - WS recalculated.
11	Process goes to sleep.
12	Process removed from main memory.
13	Process has pages removed from secondary memory.
14	Process goes to sleep whilst holding a semaphore.
15	Process has its drum working set recalculated.
16	Process page removed during process removal.
17	Process page removed from its working set.
18	Process is created.
19	Process begins its log-out sequence.
20	All traces of a process are removed.
21	Process is suspended after a partial preload.
22	Process resumes after a suspension.
23	Process has a copy of all pages it has written to backed up on the tertiary level.
24	Process undergoes an extra-strobe.
25	Virtual and physical addresses of a preloaded page.
26	Process issues a supervisor call.
27	Process has a page moved between secondary memory states.
28	Exit from the supervisor state.
29	A page is written to secondary memory.
32	Current lengths of scheduler queues (every 10 secs).
33	Monitor starts or restarts after a gap.
34	Monitor closes down.

All events <32 may be selected via a mask set at start up.
All events >32 are always switched on.

- A process completes its preloading sequence and may become eligible for the CPU (event ≠≠ 4). The number of pages preloaded and the process status (which gives knowledge of whether the preload was partial or not) form the parameters.

- A process issues a page fault, the event recorded will depend upon the level in the storage hierarchy at which the page is to be found - tertiary, secondary or primary memory (events ≠≠ 5, 6 and 8). The virtual address of the page and the corresponding physical core frame allocated to it are the parameters here.

- A preload page has arrived in main memory (event ≠≠ 25). The parameters are the same as above.

- A page faulted page eventually arrives in primary memory and is ready for use by the process (event ≠≠ 7). There are no parameters, a process may only have one page fault outstanding at a time.

- A page is to be removed from primary memory either during the recomputation of the process' working set (event ≠ 17) or when the process is being removed from primary memory (event ≠ 16). The virtual address, physical core frame and a bit mask showing how this page first came to core (demand or preload) and whether this page was read, written or unused during this residency are the parameters.

- A process has had all its primary memory allocation removed and is no longer a member of the MPS (event ≠ 12). The process status and the category to which the process is now assigned are the parameters.

- All of a process' pages have been backed up to the tertiary level in the hierarchy, perhaps freeing pages at the secondary level (event ≠ 13). The secondary memory allocation and the block page table allocation form the parameters.

b) Asychronous Process Phenomena

All of these events are non-paging events depending on the characteristics of the process. Such events may be recorded whenever:

- A process wakes up i.e. parameters arrive from a terminal and that process becomes active and competes for system resources (event ≠≠ 1). The category to which this process is currently assigned is the parameter recorded here.

- A process goes to sleep i.e. the process outputs to its console and becomes dormant awaiting a reply (event ≠≠ 11).

- A process goes to sleep whilst holding a semaphore (event ≠≠ 14).

- A process is created (event ≠≠ 18).

- A process begins its logout sequence (event ≠≠ 19) or finally has all traces of its existence removed from the system (event ≠≠ 20).

- A process issues a supervisor call (event ≠≠ 26). The parameters recorded are the identifier of the SVC and the current level of the process i.e. director or user.

- A process requests that a copy of the pages it has recently changed now be copied back to the disc (event ≠≠ 23). This event may also be issued at the behest of the scheduler, the parameter identifies where the request originated.

c) Scheduler Induced Phenomena

It could be argued that everything happening within EMAS is in some way a scheduler induced event, however events classified here are non-paging events which are dependent upon the behaviour of the system scheduling algorithms. Such events may be recorded whenever:

- A process is placed on one of the scheduler queues (event ≠≠ 2). The queue involved is the parameter. The process is considered to remain resident on that queue until another significant event takes place concerning that process.

- A process overruns one of its commodity limits either a table allocation or the resource limits imposed via the category table (event ≠≠ 9). The parameter specifies the commodity involved.

- A process has overrun its core allowance but has not yet been "strobed" (event ≠≠ 24), a recalculation of the working set is carried out to see if the working set diminishes and the process can be allowed to remain in primary memory.

- A process has reached the end of a strobe interval and an attempt is about to be made to recompute its working set (event ≠≠ 10). The parameter in this case is the CPU time still allowable to the process during this residency in 4-75 clock ticks.

- A process which has just completed a "partial preload" is subsequently suspended owing to insufficient core being available to it (event ≠≠ 21). The current core allowance and current core used by this process are recorded.

- A process which has been suspended is subsequently released to re-enter the run Q's (event ≠ 22).

- The drum working set for the process is to be recalculated (event ≠ 15). An identifier associated with the algorithm to be used (there are currently four), the secondary memory allocation and block page table allocation before this recalculation, form the parameters.

- A page belonging to any process is moved between drums (event ≠ 27). This is done to ensure all pages in the secondary level are packed onto as few drums as possible - hence denser packing, hence more prepaging efficiency and automatic migration of pages off the pseudo-drum as space becomes available elsewhere. This is only done when a page is moved into primary memory during the normal activity of the process and the secondary memory page indices involved in the move are recorded.

- Exit from supervisor state (event ## 28). A normal user process takes over the CPU at the end of a burst of supervisor activity. The current process level (user or DIRECTOR) and the current values of COREF (physical main memory pages still free) and COREL (main memory still unallocated) are the parameters recorded.

- A page is written back to immediate store (event ## 29). The parameter tells why this page is being written e.g. page creation, or all pages of a process being removed from active store.

d) Monitor Events

These three events are always active when the monitor is switched on i.e. they are not affected by the setting of the event mask. They are recorded whenever:

- The monitor is switched on or restarted after a gap caused by having run out of buffer space (event ## 33). The current value of the date and time of day as held by the system are recorded.

- The monitor is switched off (event ≠≠ 34).

- A ten second alarm clock interrupt is received (event ≠≠ 32). The current lengths of the scheduler queues and the number of users currently signed onto the system are the parameters.

It must be noted that the set of events contained in the standard monitor as described above give adequate information to reconstruct queue length distributions and wait time distributions for all the nodes in the P.M.M. graph, as well as information on the access patterns within the virtual memory.

The possible range of data obtainable on a system like EMAS is vast. This version of the monitor was never intended to be a fixed, totally general monitor obtaining every possible item of performance data that might ever be of interest. Instead the monitor provides the general mechanism through which performance data may be syphoned. The structure and comparative ease with which a new supervisor may be remade allow this flexibility. All the probes implemented in the standard monitor are contained within one of the supervisor components. However to prove the flexibility of this mechanism,

additional probes have been incorporated in at least one other component for investigations into certain specific areas [Adams et al. 1977] and a modified set of probes were used in the EMAS Performance Experiment (see next chapter).

Using the standard version all data considered to be of interest for this exercise may be obtained. It may be noted that the monitor is not symmetrical, especially in the paging events class, i.e. the obvious construct of recording a "page-in request" event and "page-here" event (on completion of transfer) and similarly a "page-out request" event and "page-gone" event is not used. Advantage is taken of the fact that preloading and removal of pages from core involve the process in a wait until the transfer of several pages is complete. From the system performance point of view only the length of that wait and number of pages involved is of interest, whilst from a process behaviour point of view the page addresses involved will also be required. The approach taken allows the greatest flexibility within the standard version, minimising the number of individual event types involved whilst giving the most flexible sets of data available. For instance, if only wait times involved in paging are required then events 3 and 4 will give timings for preloadings: 5, 6, 7 and 8 will

give timings for all demand paging; events 9, 10, 11 and 12 will give timings for the bulk move involved at the end of a core residency. If use patterns within virtual memory are required then events 5, 6, 8 and 25 give the times at which pages start being used in primary memory; events 16 and 17 the times (approximately) at which pages cease to be of use to the process. In practice it was found (as had been expected) that paging events (plus event ≠ 28) would dominate the types of events which would be recorded. The total number of paging events to be recorded during any particular monitoring session had to be kept as low as possible - hence reducing overhead caused by the monitor and lengthening the total length of time for which the monitor would run before filling its data space. In normal use the monitor in its standard form monitoring all events (except event 25), collects around 300,000 events in about twenty minutes.

Interference caused by this monitor whilst in use is approximately a four percent addition to the supervisor CPU time. The level of interference will depend upon the number of events monitored. Whilst in use the monitor claims two pages of buffer space, thus reducing the available user core space by about one percent on a one M-byte configuration. The gathering routine adds approximately 2,300 bytes of

code to the resident supervisor code, plus 3,200 bytes for organising the switching on and off of probes. The interference caused on the pseudo-drum channel is minimal with less than one page of data per second being transferred. It must be noted that none of the monitoring aids implemented at this level cause any direct interference with user processes characteristics within the virtual machine of that process.

Sampling Monitor

A monitor was also constructed to give summary information on how the system is performing by sampling certain critical variables. This involves one routine in the supervisor which is activated at regular intervals to accumulate a total, maximum and minimum observed value for each of the chosen parameters. The contents of this table is dumped, and all the values re-initialised either at regular intervals or by the setting of a system test flag. A machine readable form of this print out will exist in the main accounting log. Table 2.4 gives an annotated example of a typical print out from this monitor. The sampling interval is currently ten seconds as there is a convenient system "alarm clock" interrupt at this time. Ten seconds is of a much larger time scale than most system phenomena appearing at this level, and thus the data hopefully

Table 2.4

SAMPLE OF Q-SAMPLE DATA

EMAS 81EB DATE:08/09/75 21.20.18

QUEUE SAMPLING INFORMATION

NO. OF TIMES QSAMPLE KICKED WAS 188

ITEM	TOTAL	MAX	MIN
RUNQ1	76	4	0
RUNQ2	247	3	0
ACT STRQ	0	0	0
ACT TKEQ	23	4	0
CORE Q1	254	12	0
CORE Q2	294	9	0
CORE Q3	111	4	0
CORE Q4	37	3	0
CORE Q5	442	8	0
CORE L	4447	141	-34
CORE F	10377	153	4
CORE S	4506	72	0
ASUNUSED	192261	1231	872
AS FREE	184387	1193	835
BPTUNUSD	48996	306	226
BPTFREE	44661	281	198
PT FREE	8708	75	17
SAM FREE	22677	127	111
PARAMTAB	27943	158	121
USERS	5490	32	25

EMAS 81EB DATE:08/09/75 21.51.51

will not be too adversely affected by any periodicity in the system. The data is dumped every 1000 samples (approximately every three hours). The periods between this regular dumping of accumulated data is controlled by a system test flag. The sampling interval could be varied, but this has not been considered necessary and the monitor has been of some use as a simple informal aid in system tuning. Interference caused by this monitor is an addition of approximately 1,000 bytes in the space occupied by the resident supervisor and a negligible addition to the supervisor CPU overhead.

Category Table Transition Matrix

One of the central concepts in the EMAS scheduling scheme is that of a process category. As the transitions between categories depend upon the behaviour of the process and the categories themselves are a crude characterisation of the processes, the transition matrix of process movement between categories will provide a rough characterisation of the current workload. The limits involved in the category table will also have a definite impact upon the system performance, and thus transition matrix will be of considerable use in tuning the category table. This monitor requires

800 bytes to hold the data, and a minimal amount of code space and execution time to gather and dump it. The dumping of the transition matrix to the line printer and re-initialisation of the data space is controlled by a system test flag. Table 2.5 shows the transition matrix for a typical session. A machine readable form of this data will appear in the main accounting log.

Conclusions

The four monitoring aids described which have been implemented within the EMAS resident supervisor provide sufficient data of a very accurate form for the evaluation described in later chapters, and hopefully for other research in this field. They are very flexible, and must not be considered fixed. This applies especially in the case of the event trace monitor and the sampling monitor, which provide proven data acquisitions routes, and new events may be added to the monitoring, or some current items deleted, as necessary, limited only by the researcher's knowledge of where the item to be monitored resides. Although care has been taken to ensure that the extra overhead induced by these monitors is minimised, this has never been taken to the extreme of hand coding the monitors, and all the implementation took place in IMP in

Table 2.5 SAMPLE OF CATEGORY TRANSITION DATA

EMAS 81EB DATE:08/09/75 21.20.14

CATEGORY TABLE MOVEMENT

FROM \ TO	1	2	3	4	5	6	7	8	9	10
1	37	0	0	0	0	0	0	0	0	0
2	0	43	13	0	0	0	0	0	0	0
3	0	13	188	92	0	0	0	0	0	0
4	0	0	92	94	0	0	0	0	0	0
5	0	0	0	0	146	0	0	58	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	1
8	0	0	0	0	70	0	0	1274	8	0
9	0	0	0	0	0	0	1	7	0	1
10	0	0	0	0	0	0	0	0	0	4
11	0	0	0	0	0	0	0	324	2	0
12	0	0	0	0	0	0	0	1	0	3
13	0	0	0	0	0	0	0	5	0	1
14	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0

FROM \ TO	11	12	13	14	15	16	17	18	19	20
	0	0	0	0	0	0	37	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	315	0	0	0	0	0	0	0	0	0
	0	0	2	0	0	0	0	0	0	0
	0	0	5	0	0	0	0	0	0	0
	428	1	0	263	0	0	0	0	0	0
	2	0	0	0	0	1	0	0	0	0
	0	0	0	0	0	7	0	0	0	0
	275	6	0	64	2	0	126	0	0	0
	3	0	5	0	1	6	0	0	45	0
	5	0	1	0	0	3	0	0	19	0
	0	0	0	85	35	0	32	0	0	49
	0	0	0	0	0	0	0	0	0	0
	2	0	0	23	0	11	3	0	9	25
	2	0	0	46	22	0	3	0	0	58

keeping with EMAS philosophy. Further in keeping with the system structure, they only record data on events and entities which exist at the level of the resident supervisor, so no monitoring of file use etc. is taken here. This information can be obtained by monitoring at the level of DIRECTOR or subsystem.

None of the monitors interfere in any way with the running of user processes within their virtual memories other than adding a small amount to the total wait time experienced by the process in obtaining service from the supervisor. The level of this interference will vary directly with the amount of data being recorded, and care must be taken to collect only necessary data when planning any measurement experiment. On the issue of privacy of users, the data obtained is purely of a performance nature. The only information of interest about processes is, generally speaking, its pattern of reference within its virtual memory in terms of page addresses, and only this data is gathered. No information is gathered at this level on the contents of those pages. It is hoped that such data gathering is not considered to be a breach of privacy.

Utility programmes exist for the transfer of event-trace data to a standard EMAS file

from the data collection area on the pseudo-drum, and for the production of appropriate event mask to be used during any monitoring session. A variety of analysis programmes have been written for the reduction of event-trace data. Though these programmes share several common routines and a common kernel in many cases, they have not been brought together under one programme, such as the Data Analysis Programme on MTS, or the Statistics Generating Package on VM/370, but remain separate entities, specific to the analysis required.

Chapter 3

The need for rigour when taking measurements of systems has already been discussed. This chapter covers in more detail a controlled empirical approach to evaluation i.e. one of observing systems under conditions in which all the variables which might affect performance are fixed, or under the control of the experimenter. A measurement experiment carried out on the Edinburgh Multi-Access System is also described in detail.

One of the great disadvantages in attempting to make an evaluation of any system is the great number of possible factors which may make an impact on the observed performance. Also, subtle interactions between factors may themselves prove to have a significant effect. With interactive systems one of the most highly variable and significant factors affecting the performance of the system is the user workload. This makes any evaluation of the components of the target system, based solely upon measurements taken on the natural system (i.e. the system running with real users during normal service periods) very difficult, as it will be nearly impossible to attribute changes in performance to individual system components or to slight changes in the user workload between any

two observed periods.

An attempt to remove this factor was made in studies of CP/67 [Bard 1973] in which an evaluation of two paging algorithms was being carried out. This approach consisted of incorporating the two algorithms in the target system software and switching between them on a very short time scale, thus hoping to eliminate any differences due solely to the workload. Measurements can then be taken on the natural system and an evaluation made of the two algorithms with some confidence. However, this approach is naturally limited in its application: it will be difficult to evaluate hardware changes, or compare software algorithms which cannot co-habit with the resident supervisor without causing considerable overhead or involving changes of such a nature that switching between them may not be possible (e.g. they may maintain differently ordered queues or paging table formats).

Workload Drivers

The only factor beyond control in the natural system is that of the user workload, as system software and hardware components may be fixed. Thus if experiments are to be carried out within a totally

controlled environment, some way must be found of providing a standard workload during experimental runs. A definition of a standard workload will be useful at this point.

A standard workload in the case of interactive systems is a total workload which may be applied to the target system in which all of the components of user characteristics are completely defined, in terms of commands issued, files and programmes used, think times between commands and expected typing delays [Holdsworth et al. 1973]. Such a workload is usually defined in terms of a fixed number of pseudo-users running from a set of one or more scripts. Each script holds a representation of an interactive conversation between a pseudo-user and the target system. One or more pseudo-users may be run from each script. Figure 3.1 shows an example of a possible script for an EMAS pseudo-user. A standard workload is deemed to be reproducible if each time it is applied to the target system the activities of each of the pseudo-users remains fixed i.e. the commands issued and the time the system spends in "user wait" for that pseudo-user do not vary from run to run. In the context of the interactive conversation shown in Figure 3.2 the part above line A - B will always be

Figure 3.1

EXAMPLE OF A POSSIBLE EMAS USER SCRIPT

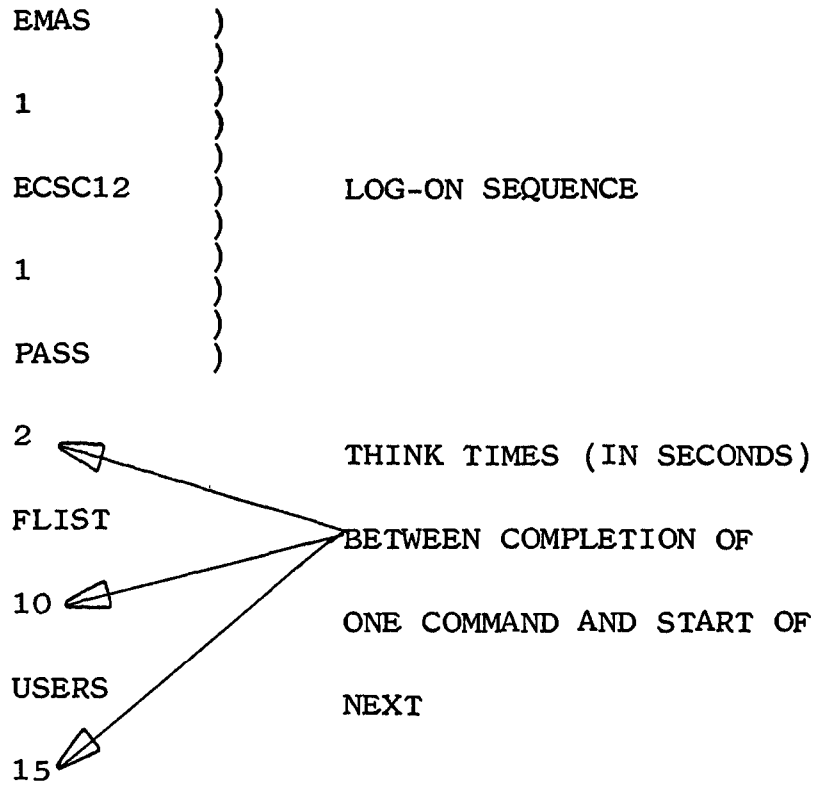
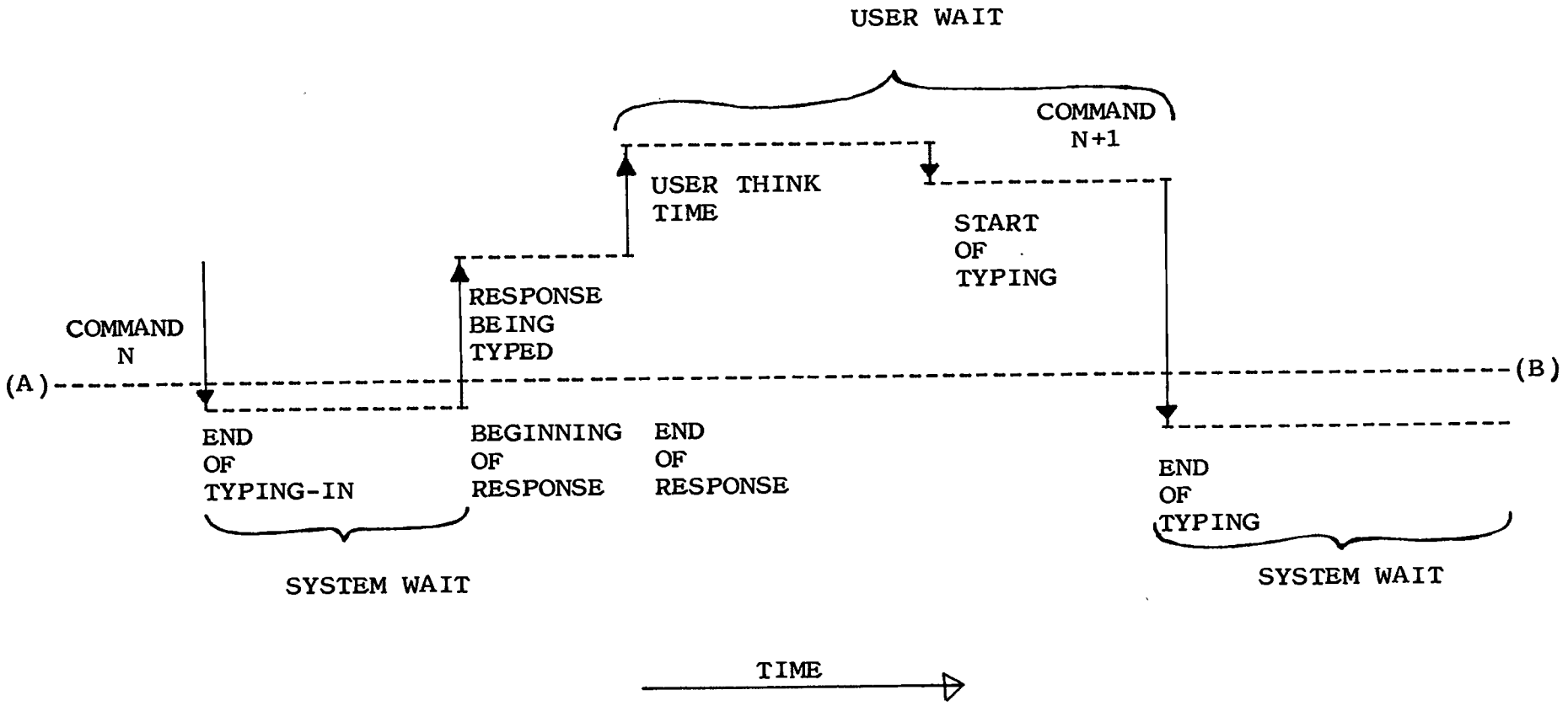


Figure 3.2

SCHEMATIC OF AN INTERACTIVE CONVERSATION



constant for that pseudo-user. During the running of the standard workload the rates at which individual pseudo-users complete their work relative to one another may, of course, vary, according to the way the system differentiates between different classes of work, but within the context of each script the user characteristics will remain fixed.

Providing a standard workload for an interactive system by running a set of batch (non interactive) jobs on the system will be unsatisfactory, as it will not incorporate any representation of user think time and will not load the communication facilities of the system in an appropriate manner. Also, on many systems it is quite possible that the full range of "interactive" commands may not be available to batch jobs. Employing a large number of humans to sit at terminals and type in commands from a prepared script defining the interactive conversation is a possibility, but it would be very tedious for the copy typists involved, and humans do make errors. It is unlikely that the workload so induced will have an exact replication of the think time and typing time distributions specified. The only way of providing a totally reproducible user workload is by a form of automatic Workload Driver (WD) or stimulator, incorporated in some system module or

piece of hardware, which will feed specified commands to the target system at the necessary times.

In the context of conversational computing a WD and the standard workload it produces should possess the following characteristics:

(a) It should load the target system by feeding specified lines of input to it, and on receiving a response, wait for a specified time (to simulate user thinking and typing) before passing in the next line of input.

(b) It must be capable of providing a relatively stable load to the target system over the measured period.

(c) It should be able to take human typing rates and terminal speeds for each simulated terminal as parameters.

(d) It should be reasonably robust and able to recover from transient errors (e.g. occasional message corruption).

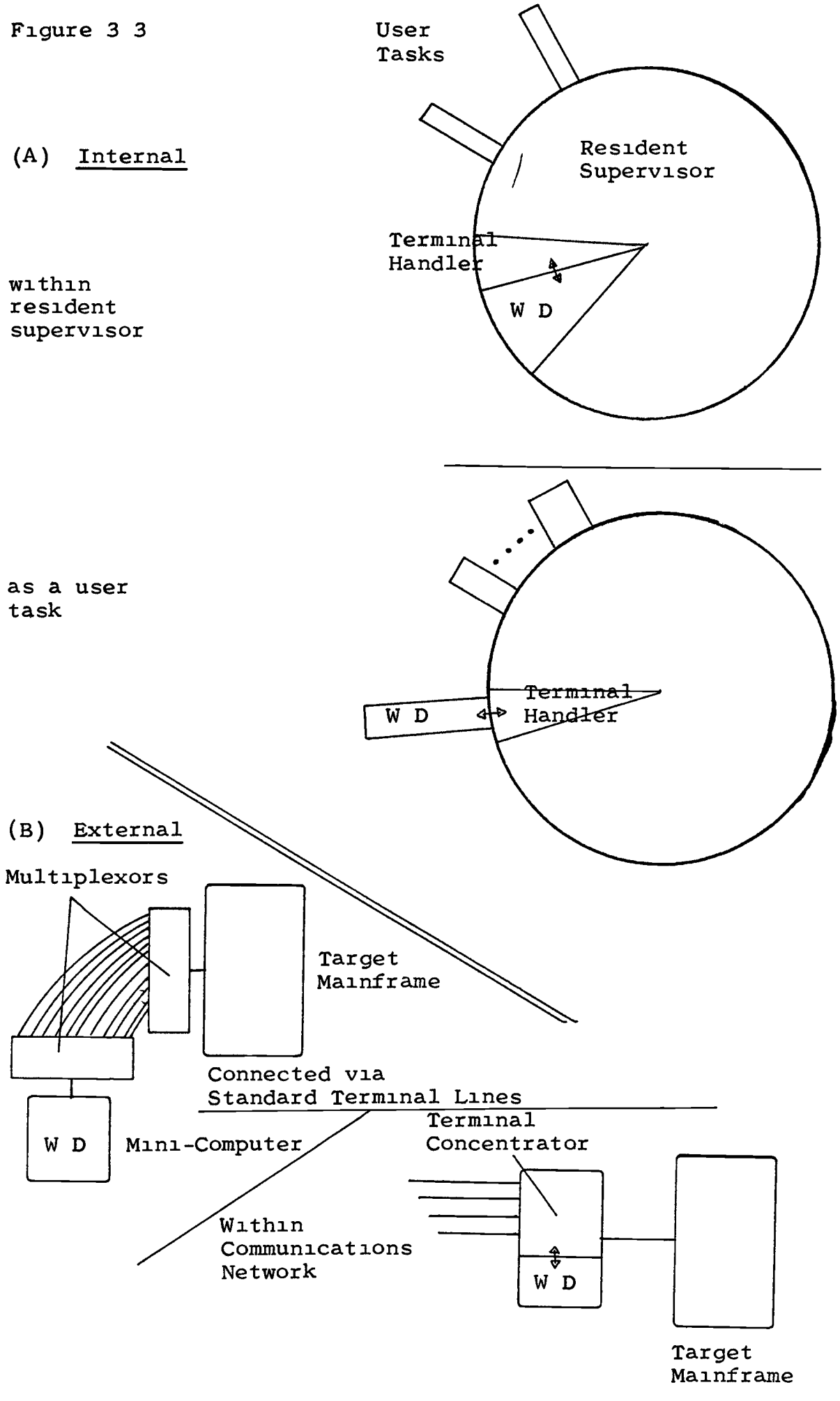
(e) It should interface easily to the target system and appear as any normal user workload would (i.e. a minimum of modification and interference to the target system).

There are two approaches to the implementation of WDs: internal and external. The internal approach is so called because the WD is incorporated within the software of the target system. The external approach involves the workload being provided from an external machine (usually a mini-computer) connected to the communications hardware of the target system.

Internal Workload Drivers

The major advantage of the internal approach may be summed up in terms of cost - no extra hardware need be provided (both methods require software), also the possibility of transmission errors causing trouble is eliminated. However, as the driver is implemented within the target mainframe, it means that it will necessarily interfere with it, consuming memory space and possibly paged I/O capacity. The WD could be implemented either as an additional user task, or within the supervisor, interfacing with the module which normally handles terminal I/O [Figure 3.3a]. This is the method adopted in the MTS - Terminal Driver Monitor [Stasuik 1976, University of Michigan 1976]. This is incorporated in the system area of the tasks virtual memory and it allows up to 200 simulated terminals to run, off up to 9 scripts. The scripts

Figure 3 3


(A) Internal
within resident supervisor
User Tasks
Resident Supervisor
Terminal Handler
W D
as a user task
W D
Terminal Handler
(B) External
Multiplexors
Target Mainframe
Connected via Standard Terminal Lines
W D
Mini-Computer
Terminal Concentrator
Within Communications Network
W D
Target Mainframe

in this case specify command lines and think times with typing delays being introduced as a simple function of the number of characters input. An internal WD is also reported to have been built for IBM's TSS/360 [Abrams et al. 1976]. This is incorporated entirely within the resident supervisor and claims to allow any number of scripts with any number of users running off each. The scripts are read in, off the card reader and presumably remain core resident throughout, which must impose some restrictions on the size and number of scripts used. Also there is no obvious way of representing user think times as such, in this case.

External Workload Drivers

An external workload driver, or remote terminal emulator (RTE), should not interfere with the target system at all, and will be connected to it via individual terminal lines, or may be attached (where appropriate) as a terminal concentrator [Figure 3.3b]. The major drawback of this approach will be the cost of the hardware in which to run the RTE. However, with the proliferation of mini-computers and the relative decrease in price of such equipment, this may not be as much of a drawback as it may first appear. Using another computer allows for much more scope in the facilities

provided in the WD, and may also extend the range of its useful tasks.

Remote terminal emulators may themselves be used as measurement devices [Abrams and Cotton 1975, Abrams et al. 1976]. Response time is the only pure performance metric by which interactive systems are judged by users. By recording and timing all messages passing between it and the target, an RTE provides a totally non interfering method of obtaining objective response time measurements. Care, of course, will have to be taken that the standard workload used whilst such measurements are taken must be an accurate reflection of the workload existing on the natural system [Barber et al. 1975]. During the development of a new system, a facility which allows a workload to be repeated time and time again may be of use in tracing system errors [Lassettre and Scherr 1972, Schwemm 1972]. In this case an external WD will have obvious advantages over attempting to implement a WD internal to a system which is itself only being developed. For systems which are in normal user service, RTEs may be used both as a tuning aid - by enabling controlled experimentation with scheduling parameters - and as a method of checking new system releases - both for performance and possible errors. By using a mini-computer and remaining completely external to the target the size and range of

scripts and workloads which may be applied will be limited only by the configuration used for the RTE and not by considerations of the interference caused in the target. External workload drivers are also more flexible than internal drivers in that they need not necessarily be system dependent (though any implementation of a standard workload will have to be) and may be used in investigations of several target systems. Several examples of remote terminal emulators now exist [Watkins and Abrams 1977].

The "STIMULATOR" facility provided by CDC on their KRONOS system [Lehmann and Gomma 1973] falls between being defined as external or internal in that it runs in a Peripheral Processor (of which there may be up to ten on the CDC 6400 series architecture) and is thus internal in that it requires no additional hardware and runs within the target mainframe, but could be considered external in that the load is being provided by a mini-computer (the PP). The "Stimulator" allows for the running of several users off any of a number of scripts and has facilities for response time measurement.

The earliest development of a true external driver was produced at project MAC at M.I.T. [Greenbaum 1969] for experimentation and testing of

CTSS and MULTICS. This RTE was based on a small (8K of store core) PDP-8 which was connected to the target systems via standard terminal lines. It could support a maximum of 12 simulated terminals running from scripts which not only contained command lines and think times, but also "verifier lines" so that the simulator can check that it is obtaining correct replies. A language was also supplied in which interactive conversations could be defined for translation into scripts. I.B.M. are also reported to have developed an external WD for testing and measuring TSO/360 during its development [Lasettre and Scherr 1972]. Unlike the project MAC stimulator this required a minimum configuration of a 360/40 and was again connected to the target mainframe by standard telephone lines. A WD based upon PDP-11 hardware has been developed by DEC [Turner 1976], one version runs in a PDP-11/20 connected to the target system via standard terminal lines, whilst another version runs as part of a terminal concentrator used on some of DEC's larger mainframes. A language to aid the definition of scripts and give some control over the parallelism of the simulated users has also been provided [Turner 1976]. External WDs are now provided commercially for some systems and may be bought or rented complete with a set of standard loads [Wright and Burnette 1976].

Evaluation Experiments

There has been some reported use of WDs in performance experiments. A set of experiments were carried out at Imperial College on the CDC KRONOS system using the CDC STIMULATOR facility [Lehman and Gomma 1973]. This involved using eight distinct scripts with up to 24 simulated users being run off each script. No attempt was made to validate the scripts, and no changes were made in the system configuration. The series of experiments (each taking more than one hour elapsed time) were run varying the maximum number of simulated terminals from 96 to 192, and all the performance data was obtained from the accounting log. One of the major performance measures taken was the time the whole standard workload requires to complete. Despite the several limitations of the experiments, considerable insight was considered to have been gained into the performance of the system and the identification of certain possible system bottlenecks.

The MTS-TDM was used in obtaining comparative data on running MTS on an IBM 370/168 and an Amdahl 470 V/6 [Emery and Alexander 1975]. The set of 45 minute long experiments was carried out using six distinct scripts with 120 simulated terminals being run

off them. The simulated terminals were not distributed evenly between the scripts, but heavily biased to some which represented particularly interactive work, e.g. editing sessions. A considerable array of data was obtained using both the DCF (an event trace monitor) and response time data from the TDM itself. This revealed several differences between the systems running on the two mainframes. During the experiments there was 5% idle on the 370 v 10-40% idle on average in the page wait state on the Amdahl, whilst though more processes were on average in the page wait state on the Amdahl, the overall paging rate was lower (more processes completing before they "lost" pages which had to be page faulted back in). Thus, though it was thought both systems were memory bound (both had 2 M-bytes), the 370 had also very little CPU to spare, whilst the Amdahl was thought to be paging bound. Response times on the Amdahl were 9·9% lower on average for a CPU which was estimated to be 50% faster.

The performance of the PDP-11/70 running DEC's resource sharing, time-sharing system (RSTS) in a transaction processing environment has been studied in great detail under simulated workload provided from a PDP 11/20 connected to the target via normal terminal lines [Kosko and Turner 1975]. In an experiment lasting nearly 12 hours, 27 simulated users

carried out over half a million transactions. As a result of this experiment, the major bottleneck was found to be in the terminal I/O hardware, and not in the disc system throughput as had been suspected before the experiment. Modifications based on this data were implemented which resulted in a 20% improvement in throughput [Turner and Kosko1976]. Such results would not have been found from an experiment driven from an internal WD. The same RTE was used in investigations of DEC's Interactive Application System (IAS) running on the PDP 11/70 [Turner and Levy 1976]. The load consisted of a mixture of jobs classified as either computational or interactive. There were up to 22 simultaneous pseudo-users during a set of 15 minute experiments when loads consisting of different mixes of these classes were run on a number of hardware configurations. The response time data obtained from the RTE was used to determine the suitability and expected performance levels of the system for various applications.

An extensive set of experiments were carried out on the Murray Hill Time Sharing System (MHTSS) at Bell Labs, using a commercially available RTE [Wright and Burnette 1976]. The interactive workload simulated was evolved from system usage data gathered on the natural system and an attempt was made to mirror

the distribution of system commands issued as well as think time and typing speeds. It is interesting to note that the think time in this case was uniformly distributed between 0 and 11 seconds compared with Scherr's observation of an exponential distribution with mean of 30 seconds of CTSS. Five distinct scripts were constructed. The stimulator allowed verifier lines to be inserted at various points in the scripts and if any error was noted from this, then the test was halted and re-run. Several simulated users were run off each script. The load varied from 30 to 90, but the ratio of the way users were apportioned between scripts remained fixed. A validation exercise was carried out running a 45 user version of the simulated workload, and comparing it with the 45 user load on the natural system. The validation was carried out at the command distribution level as well as at the deeper level of the internal supervisor queues. Minor variations occurred in the target system configuration during the time span of the experiments. A variety of monitoring aids were used, and extensive data obtained using the event trace monitor embedded in the system. The data included processor, channel and various levels of memory utilisations as well as supervisor overheads. The researchers involved held this exercise to have been very useful, and considerable insights to have been gained from the set of hour long experiments.

However, considerable effort was required, and some 187 hours of stand-alone time was used to run the experiments, one third of which were consumed in debugging or failed runs.

During the development of IBM's Time Sharing Option both a simple performance model and a remote terminal emulator were used to check out the system's performance [Lassettre and Scherr 1972]. The RTE consisted of an IBM 360/50 connected to the target via individual terminal lines. The scripts used by the pseudo-users were not deterministic as is normally the case, but consisted of a set of subsessions. Each time a pseudo-user completed a subsession the decision as to which subsession is to be executed next was taken at random, with a weighting factor which determined the overall mix of the total workload. Data collected included both response time and target system measurements from an event trace monitor embedded within TSO. This data was used in the validation and calibration of the simple performance model.

CP/40 (a predecessor of IBM's CP/67) was used in interesting experimental investigation of the influence on paging behaviour of four major factors [Tsao et al. 1972, Tsao and Margolin 1972].

These factors were:

a) Replacement algorithms.

b) Load sequence of system subroutines.

c) Main memory size.

d) Problem programmes.

In this investigation a full factorial experiment of 81 separate runs was used. The load in this instance consisted of a set of three Fortran programmes which were compiled but not executed. The data gathered, in terms of usage information on individual pages and paging events, was used in formulating empirical models of the system. Though the system used was only a uniprogramming one and the load used was very restrictive and did not use any form of WD, the methodology adopted in this investigation is very interesting.

The EMAS Performance Experiment

The EMAS performance experiment was devised to provide a consistent set of data in an investigation of the effects of various system components upon system performance. These measurements,

taken on EMAS, running in as near normal conditions as possible, using a variety of hardware configurations and two different paging algorithms, would provide the basis for an empirical evaluation of the system, and would also be used in validating and calibrating a model of it.

The parameters which were varied in this experiment were:

(a) The amount of main store available to the system. This is carried out easily owing to the highly parameterised nature of the system, by setting an appropriate system variable at Initial Programme Load (IPL), which defines the amount of primary memory the system may use i.e. no physical removal of primary memory took place. Three different values of main memory were used - 5/8 M-bytes, 3/4 M-bytes, and 7/8 M-bytes corresponding to 112, 144 and 176 pages respectively available to user processes.

(b) The number of channels available to the secondary memory (drums) and hence the paging I/O capacity. This is effected by setting a hardware switch before IPL. The system is to a certain extent self configuring and automatically checks at IPL which channels are available to it and acts

accordingly. Either one or two channels could be used.

(c) The process scheduling algorithm. The two variations on this were:

(i) Using Working Set Replacement (WSR) i.e. whenever a process is admitted to the Multiprogramming Set, prepage in its current working set, then demand page, until the process is due for removal. The normal category transitions (Table 1.1) were used with this scheme.

(ii) Using Pure Demand Paging (PDP) i.e. whenever a process is admitted to the MPS then only the master page is prepaged (it must be in main memory before the process may be given the CPU). This prepage transfer may itself be considered as a demand page fault. When running in this mode the category transitions carried out by a process are changed, in that each time the process goes to sleep then it is moved down a category (to NCY 3 instead of NCY 4).

To investigate the effects of these three factors a full factorial experimental design was adopted [Mendenhall 1968] involving 3 x 2 x 2 = 12 experimental runs. Table 3.1 shows the experiments conducted.

Fixed Parameters

All other factors which might affect system performance were kept fixed. The hardware used - CPU, channels, device controllers, disc files, drums, communication devices - was always the same (except those factors varied as part of the experiment). The software (with the exception of the variations in scheduling mentioned and one minor error corrected in DIRECTOR after four runs) was always the same. The user workload was also kept fixed using an RTE and a standard workload derived from a detailed benchmark defined by personnel of the Edinburgh Regional Computing Centre [Adams and Millard 1975]. The suitability and reproducibility of this standard workload is discussed in the next chapter. The hardware used in the experiments is shown in Figure 3.4.

Measures

The measures which would be of interest

Table 3.1

THE EXPERIMENTAL RUNS

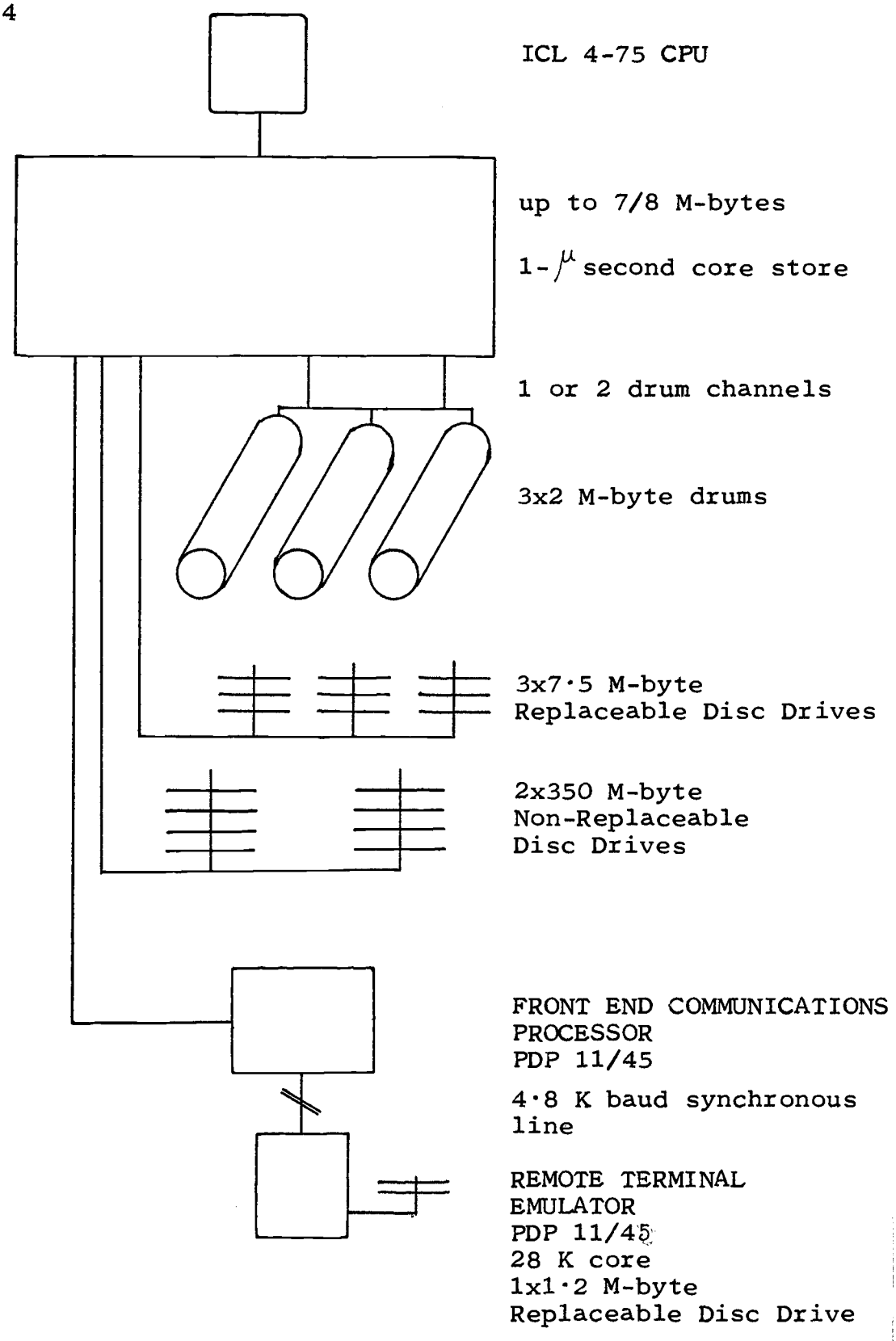
EXPERIMENT NUMBER	PRIMARY MEMORY (M-BYTES)	CHANNELS TO SECONDARY MEMORY	SCHEDULING ALGORITHM
A	7/8	2	WSR
B	7/8	2	PDP
C	7/8	1	WSR
D	7/8	1	PDP
E	3/4	2	WSR
F	3/4	2	PDP
G	3/4	1	WSR
H	3/4	1	PDP
I	5/8	2	WSR
J	5/8	2	PDP
K	5/8	1	WSR
L	5/8	1	PDP

WSR - Using Working Set Replacement Policy

PDP - Using Pure Demand Paging scheme

All experiments were carried with a fixed workload of 32 simulated users. Hardware consisted of the ERCC ICL 4-75 (machine "B" complex) with 3 drums + 1 pseudo drum. EMAS, version 814, was used throughout, as were the executive processes - Volumes version 834, Demons version 877. Runs A-D used DIRECTOR 871, E-L used DIRECTOR 872 (a minor error corrected).

Figure 3.4



HARDWARE USED IN THE EXPERIMENT

in this investigation were obtained using the EMAS measurement facilities described in the previous chapter. These included:

1) CPU utilisations.

2) Counts of various classes of paging and the wait time spent in each.

3) Times spent in each of the scheduler queues.

4) Times spent in each of the major process states.

5) Scheduling decisions taken.

6) Throughput rates.

These were obtained using the CPU time monitor and event tracing facility. The version of the event trace is a modification of the standard scheme. As the access patterns within individual process working sets are taken to be the same in each run no virtual memory addresses were recorded. Similarly, rather than recording an event each time a page is removed from main memory, only one event is recorded each time a process is strobed or removed from core showing

the number of pages involved and usage information. The events used in this version of the event trace are shown in Table 3.2. This cutting down on the number of events monitored and the parameters recorded helps to keep the overhead due to this monitor as low as possible.

The queue sampling monitor and category table trace were also used. These both induce very little overhead and produce a set of easily interpreted data which may throw further light upon the subject. The total length of any run was not taken as a measure. There are two major reasons for this. The original version of the benchmark used took approximately two hours to run on a configuration consisting of 3/4 M-bytes of core, two drums and one drum channel. To allow the experiment to run to completion would have greatly lengthened the time taken for the experimental runs. It is quite possible that a small number of jobs may be discriminated against by the system, either by the scheduling or by random placement of pages on rotating devices. These may then take a long time to complete and dominate any measure based only on total run times. This view was reinforced by the experience of the ERCC staff involved in the benchmarking exercise.

Table 3.2

TRACE MONITOR EVENTS USED IN EXPERIMENT

IDENTIFIER	EVENT
1	Process wakes up.
2	Process put onto scheduler queue.
3	Process enters Multiprogramming Set.
4	Process completes preload.
5	Process page faults - page on tertiary memory.
6	Process page faults - page in secondary memory.
7	Page faulted page arrives in main memory.
8	Process page faults - page in main memory.
9	Process overruns a category resource limit.
10	Process completes strobe interval - WS recalculated.
11	Process goes to sleep.
12	Process removed from main memory.
13	Process has pages removed from secondary memory.
14	Process goes to sleep whilst holding a semaphore.
15	Process has its drum working set recalculated.
16	Process begins removal from MPS.
17	Process has working set recalculated.
18	Process is created.
19	Process begins its log-out sequence.
20	All traces of a process are removed.
21	Process is suspended after a partial preload.
22	Process resumes after a suspension.
23	Process has a copy of all pages it has written to backed up on the tertiary level.
24	Process undergoes an extra-strobe.
27	Process has a page moved between secondary memory states.
28	Exit from the supervisor state.
29	A page is written to secondary memory.
32	Current lengths of scheduler queues (every 10 secs.).
33	Monitor starts or restarts after a gap.
34	Monitor closes down.

All tracing is turned on automatically.

No addresses are recorded in events 5, 6, 8, 16 or 17.

Workload Driver

The remote terminal emulator used was implemented on a 28K PDP 11/45 by personnel of the ERCC [Gilmore and McBride 1975, Gilmore 1976]. Owing to limitations in the hardware used, this could maintain a maximum of 32 pseudo-users at any one time. Each pseudo-user ran off its own private script. Unfortunately there was no way of logging messages between the RTE and the target, so this was only used as a method of producing a reproducible standard workload, and was not used as a measurement device. Thus all measurements taken in the experiment took place in the software of the target machine. The PDP 11/45 was originally connected to the original, hardwired communications multiplexor (MCCCU) via a 4·8 k-band synchronous line, but was later connected via a Front End Processor (formed by a PDP 11/45) which was in turn connected to the 4-75. At all times the RTE was connected as a terminal concentrator. An alternative to this approach would have been to implement a workload driver internal to EMAS itself. This could possibly have been done by placing the driver in the resident supervisor module which handles the interactive communications hardware. Owing to the hierarchical design of EMAS this would probably have meant that all the scripts would have had to be kept

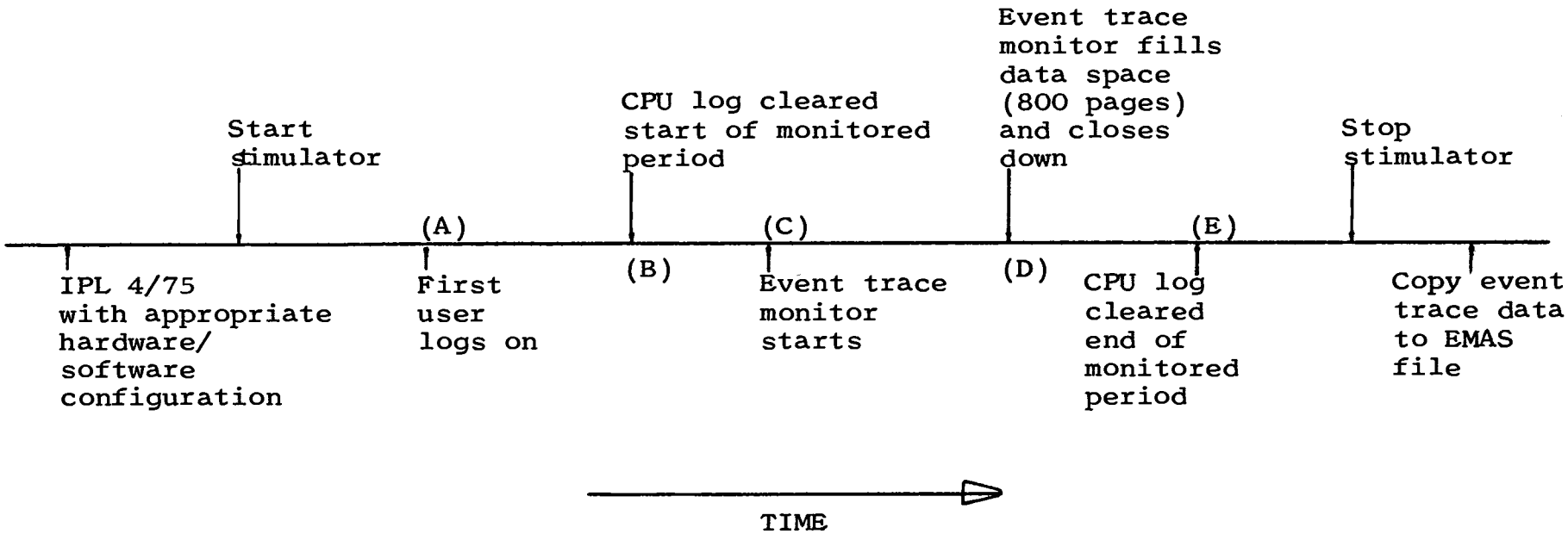
in main memory as it would have been extremely difficult for the internal W.D. to access files. The interference which such a driver would have caused was another factor of concern, so this approach was not taken.

Experimental Runs

Each run included in the experiment took the following format (Figure 3.5). First EMAS was IPL'd with an appropriate hardware configuration. The RTE was then started and pseudo-users proceeded to log on. The timing of the run started with the first pseudo-user logging on. Eight minutes after the start the tables for the CPU log queue sampling and category transition trace were cleared and a 31 minute measurement window began. The eight minutes was more than adequate time for all users to log on and the system to achieve a form of steady state. Two minutes after the start of the measurement window the event trace monitor was turned on and it continued to gather data until it had filled its available data space (800 pages) and switched off. At the end of the measurement window the CPU log queue sampling and category transitions data were dumped. This marked the end of the run and the RTE was closed down. The event trace data was then retrieved and stored in a standard EMAS file for later analysis. EMAS was then closed down and re IPL'd for the next

Figure 3.5

A NORMAL EXPERIMENTAL RUN



A⟶B	6 minutes	
B⟶C	2 minutes	
B⟶E	31 minutes	(monitored period)

The close of the event trace monitor (D) will normally take place before the end of the monitored period (E) when the data space is filled. If not then the event trace monitor is shut down at (E)

run. All the switching off and on of monitoring facilities took place automatically. Any interference caused by the monitoring should be nearly identical for each run as the amounts of data obtained (the interference being roughly proportional to the amount of data taken) was the same in each case.

Exclusive use was required of the mainframe during these experiment runs. During the spring of 1975 a daily slot was provided by the ERCC management in the early mornings in a period normally taken up by system housekeeping functions (archiving etc.) for a series of runs. At this time the PDP 11/45 was connected via the MCCCU. After a number of runs had been completed a fault was found in the EMAS software handling the buffering of messages to and from the RTE. This fault was considered to have had a drastic impact on the results obtained, so all that data was abandoned. However, the experience gained in running procedures and proving the software in the RTE was very valuable.

During the summer vacation of 1975 the ERCC allowed one 4/75 configuration to be taken out of service at 8.00 p.m. on certain evenings, and given over to the experiment. A total of 13 evenings were dedicated to this with approximately three hours per

evening being available. The first six evenings were taken up checking that everything worked according to plan i.e. that the RTE worked, that all the scripts would run successfully, and that the level of loading was adequate. There was still one process failure during experiments A - D, approximately 15 minutes into the monitored period. From these failures an error was found and corrected in DIRECTOR. The effects of this change on the system performance was not considered significant (other than removing the failure). Any experiment runs which suffered from any major system failure, either in the target or the RTE (there were hardware failures in both) were discarded and re-run. An exception to this is Run 'F' which suffered from a hardware failure after the end of the measurement window. This meant the CPU log was lost but the trace data was safe. A further run on this configuration was planned, but could not be carried out as the hardware for the RTE was moved to a different site.

Conclusions

The validity of the workload used in the Performance Experiment is discussed in the next chapter, and results obtained will also be given in succeeding chapters. The conclusions drawn from the

experience of carrying out such an exercise and the feasibility of adopting such an approach are:

(a) The initial setting up of the framework (i.e. debugging the WD-target system interface hardware and software) in which such experiments may be run is tedious and is fraught with a large number of possible sources of error and frustration. Once such a framework is proven, work can progress at a reasonable rate (this would also appear to be the experience in the M.H.T.S.S. evaluation).

(b) The time required to run any extensive set of experiments plus the initial setting up phase will be quite long, though there are, of course, well proven experiment designs which allow for a reduction in the number of actual runs which need be carried out [Cochran and Cox 1957] and one need not carry out a full factorial experiment as was the case here. An enlightened attitude will be required by the system management to allow dedicated time required for such an exercise. Once the initial troubles had been ironed out it took almost exactly one hour per successful experiment run.

(c) If a similar approach to that carried out here is adopted, and an extensive set of event trace data (800 pages per run approximately) is taken, then a non trivial data management problem results. A reasonable procedure for handling this data efficiently must be worked out in advance. During the performance experiment the data from each night's run was analysed as soon as possible during batch runs that evening (using three EMAS processes), and archived to magnetic tape immediately thereafter for possible re-use at a later date.

(d) In terms of pragmatic approach to system evaluation a development of this method must be seriously considered as an essential route for further work, so that some form of data base of empirical data on such systems can be built up for reference. The standard workload used here, and the stimulator used, were by no means ideal examples of their kind, but the modified interactive benchmark did provide what must be considered as a very reasonable representation of a natural workload for such a system (see next chapter). Similarly the RTE presented this standard workload to the target system in a realistic fashion. It was considered best to use these tools which were available and in which some experience had already been gained rather than starting completely clean and

repeating to a great extent the large amount of work which had already gone into the benchmark definition and stimulator construction. Based on the experience gained from this work an improved RTE has been designed and implemented [Adams et al. 1977]. The area of workload definition and experimental procedures is an area in which much further work could still be invested.

Chapter 4

In this chapter the workload applied to the system during the EMAS performance experiment is examined in greater detail. Some measures of workload at the level of the EMAS resident supervisor are given and the reproducibility of the standard workload in terms of these measures is discussed.

Benchmark Construction

The standard workload used in the experiment was derived from an interactive benchmark defined from measurements taken on EMAS at the level of the standard subsystem [Millard et al. 1975, Adams and Millard 1975]. This involved recording such items as the distributions of: the types of commands issued, sizes of files used, user think times, system resources used by individual commands (page transfers and CPU times) and the length of interactive sessions. A PDP-8 interposed between certain terminals and the normal communications hardware was also used to monitor user typing characteristics in greater detail. This measurement of user behaviour was carried out over a lengthy period, and the workload was found to be quite stable in terms of the items monitored.

A benchmark was then defined in terms of 32 distinct scripts, each script describing a two hour session at a pseudo-terminal. During each of these two hour sessions, several pseudo-users in turn made use of the pseudo-terminal. The scripts interacted with a set of base files in the target system. These files were never destroyed or modified in any way, all use of them was carried out in terms of operations (e.g. editing) from the base file to some temporary file which was subsequently destroyed. Though EMAS allows users to type ahead - i.e. give input to the system before a request for input is issued - no attempt to simulate this was made in the RTE, each think time started from when a reply was received by the RTE.

The benchmark was validated in terms of supervisor activity by measuring the CPU utilisations and induced paging rates over the two hour period and checking this against measurements taken on the natural system. The RTE used was validated in terms of accuracy in reproducing the scripts by monitoring EMAS again at the subsystem level whilst the benchmark workload was being run. This measurement log was then checked against the scripts used. All of the benchmark definition and validation was carried out by ERCC personnel for system acceptance trials.

It was decided to base the standard workload used in the experiments on this benchmark because so much work had already been invested in its specification and validation, and it was considered to give an acceptable representation of a realistic user workload [Millard 1975]. The characteristics of the workload at the level of the resident supervisor will be of interest in the experiment. It would have been possible to interpolate the workload at this level by using a set of synthetic programmes rather than relying on a benchmark which was defined at a higher level. However, to obtain a workload which mirrored real user behaviour in a realistic fashion would have involved an effort at least as great as that put into the benchmark construction. Not only is the distribution of process working sets quite wide but EMAS processes will make use of a variety of subsystem and DIRECTOR facilities during their execution. Their working set composition and especially the level of sharing of pages will reflect this. It may have been difficult to obtain a reasonable distribution of such requests and working set sizes in a small set of synthetic programmes. Also, as is shown later, any workload defined in terms of activity monitored at the level of the resident supervisor will be in some ways dependent upon the algorithms employed therein. If such a workload is then to be used as a basis of

comparison for two algorithms employed at this level, great care would have to be taken that the parameters chosen to define the workload were in no way influenced by these algorithms.

Modifications to the Benchmark

The standard workload used in the EMAS performance experiment was a modification of the benchmark in the following ways:

- The length of run used was an eight minute period (to allow all users to log on - a ten second gap was required between pseudo-user log-ons to avoid overloading the RTE - and settle to a steady workload), plus a 31 minute monitoring window (30 minutes was the observed mean length of a user session). Thus only the first 39 minutes of the benchmark was used. The workload was considered to be spread evenly throughout the two hour period [Millard 1975]. Also the base files were never tampered with and the temporary files always destroyed before being reused, so there should be no interference from files being lost due to cutting the benchmark short. By doing this, the time taken for each experimental run was considerably reduced and could each be carried out in approximately one hour.

The benchmark was defined and validated on an EMAS configuration consisting of 3/4 M-bytes of core store, two 2 M-byte drums and one drum channel. The 32 simultaneous users represented in the scripts provided such a configuration with a reasonable level of loading. However, as more powerful configurations were to be used in the experiment, a higher level of loading would be required if the target was to be in a heavily loaded state over the observed period. Two ways of achieving this were considered:

a) Increase the number of scripts and simultaneous pseudo-users.

b) Decrease the think times in the current scripts.

The latter approach was adopted as limitations within the RTE hardware made it impossible to increase the number of simultaneous users significantly, and it would have required a further validitation process to check that the command distribution presented by such a new benchmark did not vary significantly from the original. The think times specified in the original scripts were modified in the RTE software according to the formula:

$$\text{THINK TIME USED} = \text{maximum } (0, \frac{\text{SCRIPT THINK TIME}}{2}) \text{ seconds}$$

This maintained the same distribution of user commands issued, keeping the target system under a reasonably heavy load throughout the monitored period, and removing any periods of idle which had appeared on larger configurations in which only one or two users of the 32 were active at any time. This was considered valid as the aim of the experiment was to compare the effect of the three chosen factors upon the performance of a heavily loaded system. The effect of cutting the think times will be to keep more processes in the active state (awake) at any time, thus giving a similar effect to having a larger number of users logged-on. The characteristics of processes in main memory will not be altered, the main difference between this increased workload and a true increase in the number of users will be in secondary memory utilisation and distribution of files used over the surface of the discs used for tertiary memory. The files used by the benchmark were held on two otherwise unused quarters of the disc file. To avoid the possibility of bunching of files on any particular area of the disc file, and thus in effect speeding up the disc accesses, owing to a decrease in head movement, a change was made in the DIRECTOR used in the experiments to scatter newly created files over the cylinders of the

disc.

Whilst any experimental run is in progress certain classes of work may be discriminated against in some way by the scheduling algorithms or by the chance positioning of rotating memory devices. The actual workload being processed during the measurement window will not always be absolutely identical. However the total workload presented to the system will always be the same so any effects of this nature are part of the way in which the system reacts to the workload. The measurement window is intended to be sufficiently large and starts soon enough after the start of the standard workload to minimise effects due to this.

Workload Measures

In a system of this type, the workload, as it may be observed at the level of the resident supervisor, will be related to the system just as the performance is related to the workload. The behaviour of each process will be characterised in terms of the reference patterns it produces i.e. the virtual memory addresses it accesses and the CPU times it expends on each page. The working set concept is a representation of this. Unfortunately exact working

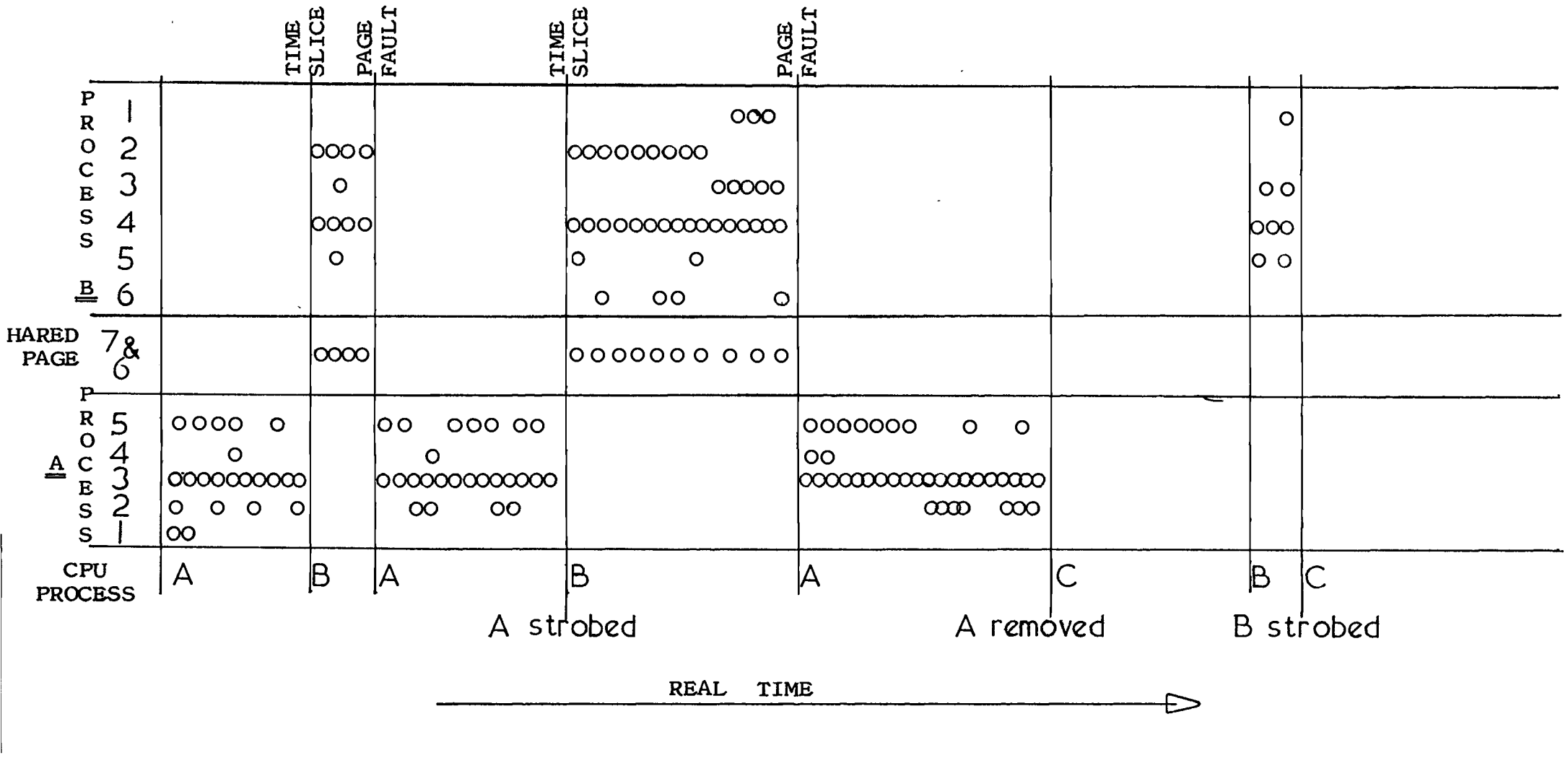
set behaviour is very difficult to measure. In the following section measures whereby the workload passing through the EMAS resident supervisor may be characterised and quantified are considered.

All process virtual memory scheduling within the EMAS resident supervisor takes place local to each process, and takes no account of the global level of loading existing on the system. The behaviour of a process whilst executing any particular interaction will thus always be the same in terms of CPU usage and paging behaviour (virtual memory accesses) and will be exactly reproducible during any two successive runs of that process. The only area in which the global system load is taken into account is in the scheduling of secondary memory space.

Within the scheduling of virtual memories of main memory resident processes there is one possible source of error. This involves shared pages. Usage information upon which working set calculations are based comes from markers associated with physical core pages rather than the process paging tables. The reading of this usage information and clearing of these markers is a time consuming procedure (there are eight markers per 4096 byte page on the ICL 4-75) and therefore does not take place at the end of each process

interval on the CPU (which would give the same effect as having marker associated with the process paging tables), but only takes place after regular intervals of process CPU time (a strobe interval) or when a process is to be removed from main memory. Consider the following scenario (Figure 4.1): Processes A and B are sharing a particular page. Process A ceases using the page early during its residency. It is strobed, but as the page has been used during this residency it remains in A's working set. The usage marker on the shared page has now been cleared. Process B now uses the page, but is suspended for some time (perhaps on a page fault to a slow disc). Process A is now removed from core and its working set recalculated. The usage markers will show the shared page to have been used and it will be included in A's working set. The usage marker is cleared. Process B is given the CPU again but is strobed before it can use the shared page again. The shared page will not be included in B's working set. It is possible to obtain an estimate of the level of this interference from usage information associated with events 16 and 17 of the event trace monitor. When running under a demand paging scheme every page brought into main memory by a process will be used. However, when they are removed from main memory some pages are marked as never having been accessed - due to this interference. Discounting

Figure 4.1



o = ACCESS TO A PAGE

pages which have been removed by the normal strobing procedure this interference would appear to be less than between ·7% and 3·5% of all pages brought into main memory.

If the EMAS scheduling algorithms remain constant and there is no contention for secondary memory space, then the activity (paging and CPU time) caused by a process during any interaction will remain the same no matter what other processes exist on the system. The scheduling decisions taken by the supervisor in handling this process whilst resident in main memory will also then be fixed. In particular the number of residencies in main memory necessary to carry out any item of work will be fixed, as will the categories used by that process. Measures which may be used to quantify the workload under these circumstances may then include:

a) Resources requested by each process during a main memory residency.

b) Scheduling decisions taken about that process during a residency.

a) The resources used by a process during any

residency have two major components:

i) Virtual Memory accesses.

ii) CPU time.

i) Virtual Memory accesses may be roughly quantified in terms of the number of pages requested or brought in during any residency. Pages may be brought in either by prepaging or demand paging - the split between these two forms of paging in will depend upon the working set calculation algorithm's estimation of the process locality (an attribute of that process alone). Pages may also be transferred in from any level in the storage hierarchy. The ratio of the numbers of pages coming from each level will be dependent upon the global mix of processes existing in main memory (reflected in the number of pages found in main memory from sharing) and the contention for secondary memory (reflected in the number of pages moved back to the tertiary level by the drum working set algorithms and hence the number of pages brought back in from the tertiary level). The fraction of pages brought into main memory which are subsequently written back to secondary memory will depend upon both the activity of that process (the fraction of pages which it writes to) and the number of pages brought in from the tertiary level (and need to be

copied to secondary memory).

ii) The total CPU time used by a process during each residency will be a function of that process' activity alone. Similarly the mean time between page faults will be a function only of that process and not of the global system load as is the case in most other systems of this class [Sekino 1972].

b) The scheduling decisions taken in relation to any process during any residency will always be the same. In particular the reason for terminating each residency - because the process has gone to sleep or overrun a resource allocation (CPU time or main memory allowance) will be a function of that process' activity. Hence the categories used by a process in carrying out its work will also be fixed.

Reproducibility of Standard Workload

The ideal way of estimating the reproducibility of the standard workload would be to run it on two or more separate occasions into identical hardware/software configurations, obtain a set of measures and compare these to obtain bounds on likely errors induced by variations in the standard workload (caused by random positioning of rotating memory

devices at start up). Unfortunately though such a run was planned there was not time to carry it out as the hardware used for the RTE was only available for a very limited period.

In the following the characteristics of the workload are presented which show that the standard workload was reproducible. This data is all obtained from the event trace monitor. The user CPU times obtained from this data (which are meant to be used to compare different runs) will be an overestimate owing to the supervisor using CPU time before the event it recorded. The error is introduced because CPU time used by the supervisor before the event is dumped is accredited to the user process. This only happens when interrupts - device interrupts, page faults or supervisor calls - occur and a simple correction can be made for this if the time used by the supervisor in servicing these is known. Whether or not such a correction has been made will always be specified when CPU timings are presented. The runs are also split into those using working set replacement (group one) and those using demand paging (group two) as certain of the measures given are dependent upon the scheduling algorithms.

The first measure of workload considered

is that of resources used each time the process enters the awake state (which is analogous to an interaction) - Table 4.1. The runs in the two groups are very consistent. One major difference between the two stems from the fact that the runs involving demand paging required an extra 0·6 residency on average per awake. This may be due to the different category scheduling used in the two groups (in group two processes are placed in NCY3 rather than NCY4 when they go to sleep). One criterion which may be used to compare the activity of process between all runs is the amount of CPU time the process consumes during any interaction i.e. how far the process moves in its computation during each interaction. There is no statistically significant difference in the corrected CPU times shown. The frequency distributions of paging requests and CPU times per interaction are shown in Figures 4.2 and 4.3 respectively, note the heavily skewed nature of the CPU distribution. The resources used per residency (Table 4.2) further show the stability of the workload.

The profile of work passing through each category in terms of residency periods page-in requests and CPU time is shown in Tables 4.3, 4.4 and 4.5 respectively. Not only do these figures show the reproducibility of work passing through each category,

Table 4.1

Average Resource Requirements per Interaction

EXP RUN	MILLISECONDS: CORRECTED CPU TIME	MILLISECONDS: UNCORRECTED CPU TIME	PAGE-IN REQUESTS	MAIN MEMORY RESIDENCIES
A	331	354	50	1·5
C	338	421	53	1·5
E	328	392	52	1·5
G	345	388	53	1·5
I	324	395	52	1·5
K	322	404	52	1·5
B	310	384	52	2·1
D	328	508	52	2·1
F	323	455	53	2·1
H	322	461	52	2·1
J	335	499	52	2·1
L	308	577	51	2·1

Figure 4.2 (a)

PAGE TRANSFERS PER INTERACTION
EXPERIMENT A (WSR)

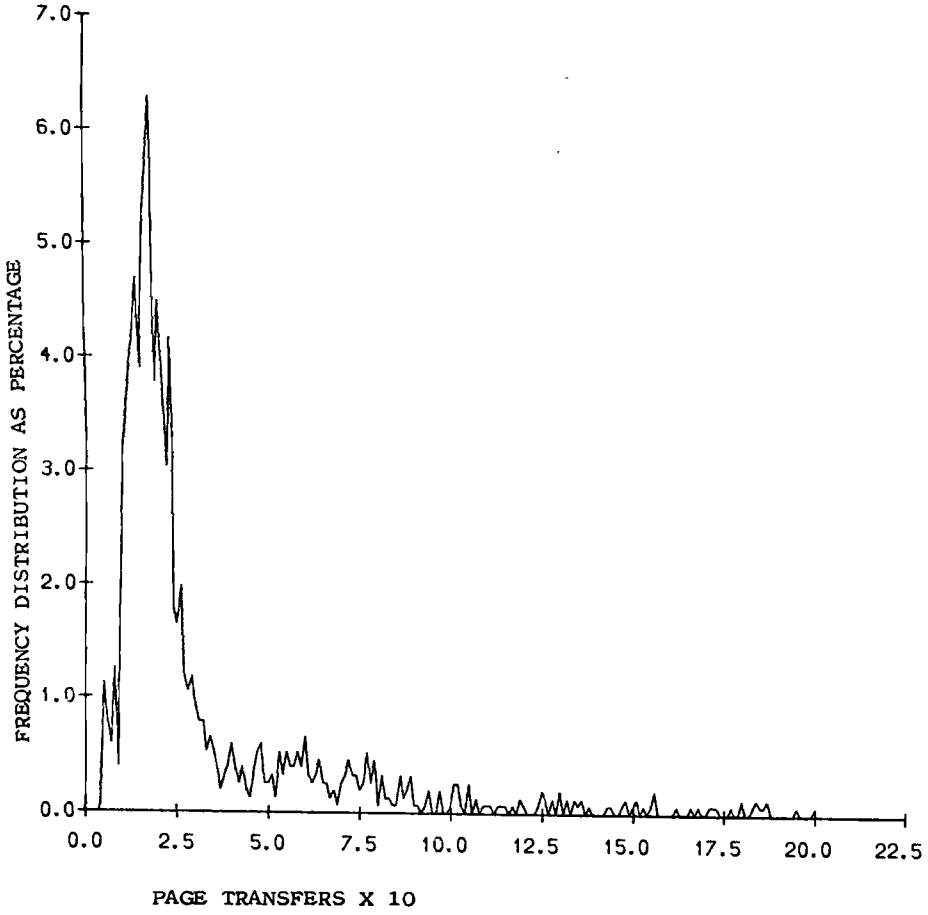


Figure 4.2 (b)

PAGE TRANSFERS PER INTERACTION
EXPERIMENT B (PDP)

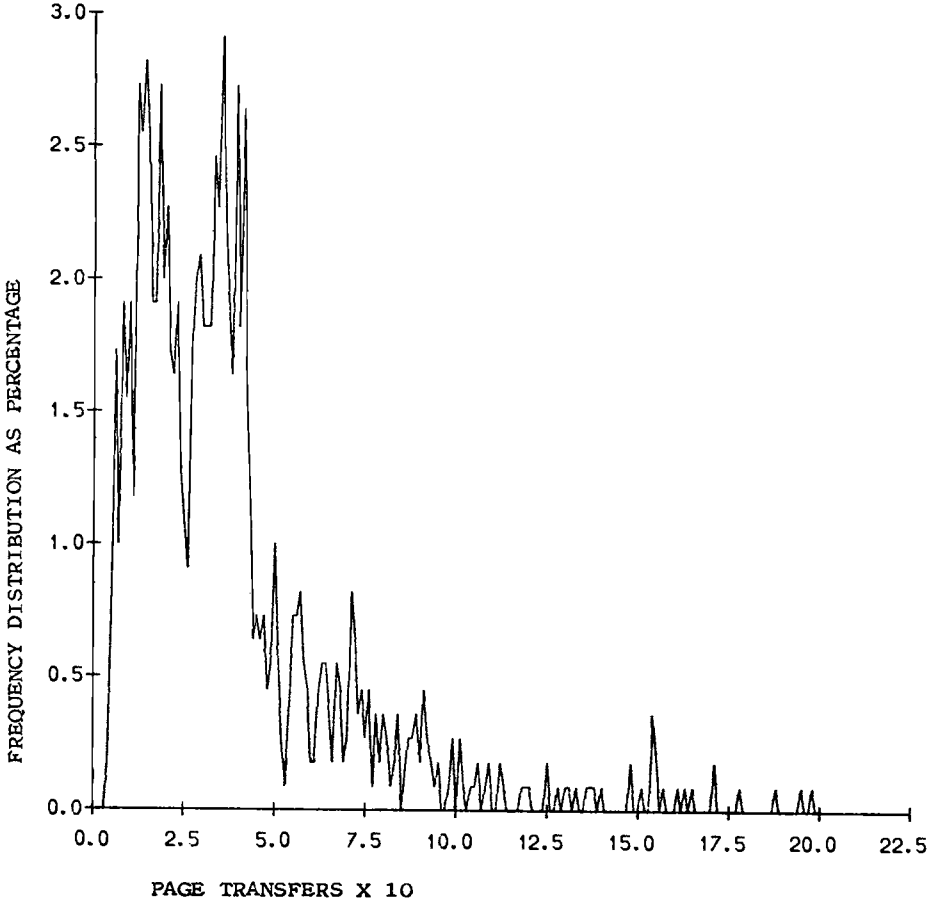


Figure 4.3 (a)

CPU TIME USED PER INTERACTION

EXPERIMENT A (WSR)

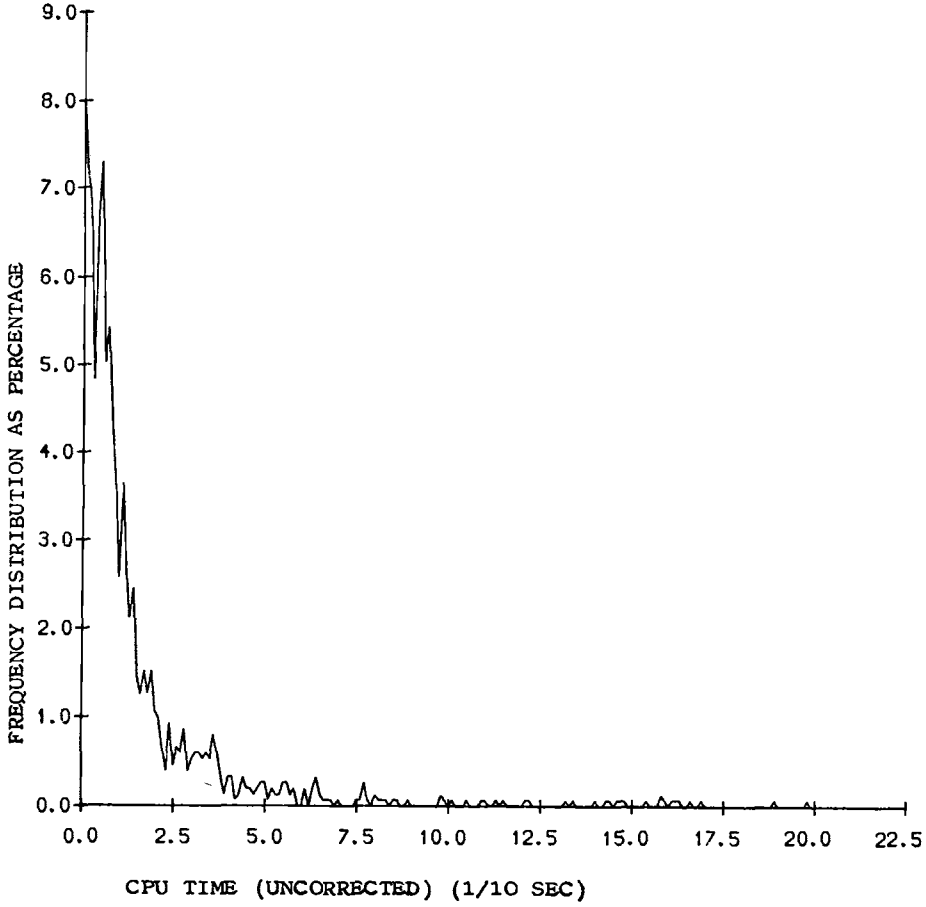


Figure 4.3 (b)

CPU TIME USED PER INTERACTION

EXPERIMENT B (PDP)

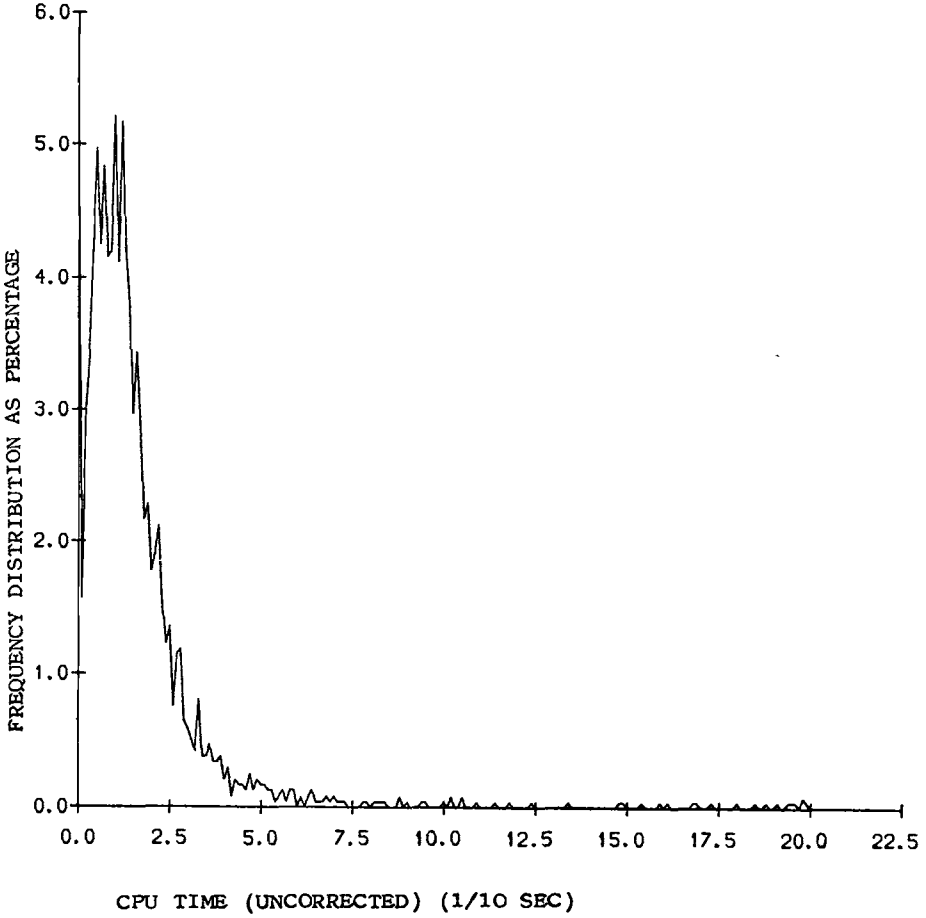


Table 4.2

Average Resource Requirements per Main Memory Residency

EXP RUN	MILLISECONDS		PAGE-IN REQUESTS	WRITE-OUT REQUESTS
	CORRECTED CPU TIME	UNCORRECTED CPU TIME		
A	224	281	35	11
C	225	304	35	10
E	219	289	35	10
G	223	341	36	11
I	217	302	35	11
K	216	322	35	10
B	146	231	26	9
D	152	260	25	9
F	150	242	25	9
H	147	293	25	9
J	159	271	25	9
L	147	317	25	9

Table 4.3

CATEGORY PROFILE - AS PERCENTAGE OF ALL MAIN MEMORY RESIDENCIES

EXPERIMENT RUN

CATEGORY	A	C	E	G	I	K	B	D	F	H	J	L
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	1	1	1	1	3	2	3	4	4	4
3	5	5	8	5	5	5	4	4	4	4	4	4
4	4	3	4	3	3	3	2	2	2	2	1	2
5	4	3	3	4	3	4	22	22	21	20	21	23
6	NU	NU	0	NU	NU	NU	NU	NU	NU	NU	NU	NU
7	NU	NU	0	0	0	0	NU	NU	NU	NU	NU	NU
8	37	37	35	36	38	38	28	28	29	29	29	28
9	0	0	0	0	0	0	0	0	0	NU	0	0
10	0	0	0	0	0	0	NU	NU	NU	NU	NU	NU
11	23	24	22	21	23	22	18	17	17	18	17	15
12	0	0	0	0	0	0	0	0	0	NU	0	NU
13	0	0	0	0	0	0	NU	NU	NU	NU	0	NU
14	11	11	10	11	10	11	10	10	8	9	9	9
15	1	1	1	1	1	1	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	NU	0	0
17	4	4	4	5	4	4	4	4	3	3	4	4
18	0	NU	0	NU	NU	0	1	0	1	1	1	0
19	1	1	1	1	1	1	0	0	0	0	0	0
20	2	2	2	3	2	2	3	3	3	4	4	3

Table 4.4

CATEGORY PROFILE AS PERCENTAGE OF ALL PAGE-IN REQUESTS

EXPERIMENT RUN

CATEGORY	A	C	E	G	I	K	B	D	F	H	J	L
1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	0	2	2	3	3	4	3
3	5	5	7	5	5	5	4	4	4	4	4	5
4	5	3	5	4	4	4	2	2	3	2	2	3
5	2	2	2	2	2	2	15	16	15	14	15	16
6	NU	NU	0	NU	NU	NU	NU	NU	NU	NU	NU	NU
7	NU	NU	0	0	0	0	NU	NU	NU	NU	NU	NU
8	29	29	28	28	30	30	26	26	27	27	27	27
9	0	0	0	0	0	0	0	0	0	NU	0	0
10	0	0	0	0	0	0	NU	NU	NU	NU	NU	NU
11	25	26	24	23	25	25	19	20	20	20	19	17
12	0	0	0	0	0	0	0	0	0	NU	0	NU
13	0	0	0	0	0	0	NU	NU	NU	NU	0	NU
14	15	15	14	15	14	15	12	12	10	11	10	11
15	2	2	2	2	2	2	0	0	0	0	0	0
16	1	1	1	1	1	1	0	0	0	NU	0	0
17	8	8	8	9	9	8	7	7	6	6	6	7
18	NU	NU	0	NU	NU	1	2	2	2	3	2	2
19	3	4	4	4	3	3	1	2	2	1	1	2
20	3	3	3	4	3	3	5	3	6	7	6	5

Table 4.5

CATEGORY PROFILE AS PERCENTAGE OF ALL USER CPU

EXPERIMENT RUN

CATEGORY	A	C	E	G	I	K	B	D	F	H	J	L
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	1	0	1	1	1	1
4	1	0	1	1	0	1	0	0	0	0	0	1
5	0	0	0	0	0	0	2	2	2	2	2	3
6	NU	NU	0	NU	NU	NU	NU	NU	NU	NU	NU	NU
7	NU	NU	0	0	0	0	NU	NU	NU	NU	NU	NU
8	8	8	8	7	9	9	7	7	8	7	7	8
9	3	0	0	0	2	2	0	0	0	NU	0	0
10	4	3	3	4	0	4	NU	NU	NU	NU	NU	NU
11	6	6	7	5	7	6	6	6	6	6	6	5
12	3	4	2	8	6	4	0	1	0	NU	5	NU
13	3	2	2	1	2	3	NU	NU	NU	NU	0	NU
14	4	4	4	4	4	4	4	4	4	4	3	5
15	28	27	26	24	25	20	6	1	5	2	2	4
16	5	9	6	10	10	8	1	5	0	NU	7	5
17	11	12	14	12	13	14	11	11	14	13	13	15
18	NU	NU	1	NU	NU	1	27	23	23	31	22	24
19	10	11	11	11	8	10	16	19	18	15	13	16
20	5	5	6	6	6	5	10	12	11	13	11	7

but also present in more detail the spectrum of work existing on the system. Within each of the two groups there is a very consistent profile with the majority of residencies passing through the interactive categories (5, 8, 11, 14, 17) - between 74% and 82%. These categories also account for the vast majority of all page-in requests - 77% - 81%. However, the CPU time used by these categories is much less - between 28% and 34%. The reasons for terminating any residency - process going to sleep or overrunning a CPU limit - are shown in Tables 4.6 and 4.7 - all other residencies are ended because of overrunning main memory limit. Again, these show a great level of consistency in all the heavier used categories. It also shows that the CPU limit has very little impact on the scheduling of processes and that when processes are removed from main memory when they have not yet completed their available work (i.e. have not gone to sleep) this is most likely to be because of overrunning a core limit. With respect to the classification of processes by the system into categories, it may be seen that this is at least partially successful in as much as those categories which have higher CPU limits are also those categories which show a higher percentage of residencies ending because the CPU limit has been reached, whereas those categories with low CPU limits very rarely, if at all, have processes rescheduled

Table 4.6

PERCENTAGE OF RESIDENCIES ENDING BECAUSE PROCESS WENT TO SLEEP

EXPERIMENT RUN

CATEGORY	A	C	E	G	I	K	B	D	F	H	J	L
1	0	0	0	0	0	0	0	0	0	0	0	0
2	86	77	67	74	72	73	29	30	36	43	43	44
3	65	65	69	71	65	65	52	45	54	52	63	47
4	99	98	98	98	98	99	96	100	84	100	95	91
5	57	57	53	50	52	54	15	13	14	10	14	14
6	NU	NU	0	NU	NU	NU	NU	NU	NU	NU	NU	NU
7	NU	NU	0	0	0	0	NU	NU	NU	NU	NU	NU
8	81	79	80	82	79	80	65	64	63	62	63	67
9	67	88	88	100	70	70	100	100	100	NU	67	100
10	0	0	0	0	0	0	NU	NU	NU	NU	NU	NU
11	62	63	60	57	62	60	52	54	56	54	57	55
12	25	20	0	0	0	0	25	0	50	NU	0	NU
13	50	30	38	72	63	60	NU	NU	NU	NU	100	NU
14	67	65	67	62	65	65	58	53	60	59	58	54
15	6	8	9	3	5	6	25	0	20	0	0	33
16	14	32	25	12	11	13	100	50	100	NU	60	67
17	45	45	45	49	49	48	58	61	37	51	45	53
18	NU	NU	0	NU	NU	0	4	9	7	0	8	5
19	38	39	37	29	39	41	7	4	5	9	6	6
20	32	38	33	34	34	40	43	38	43	36	42	49
OVERALL	68	67	67	65	67	67	47	46	47	46	48	48

Table 4.7

PERCENTAGE OF RESIDENCIES ENDING BECAUSE PROCESS OVER RAN CPU LIMIT

	EXPERIMENT RUN											
CATEGORY	A	C	E	G	I	K	B	D	F	H	J	L
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0
6	NU	NU	0	NU	NU	NU	NU	NU	NU	NU	NU	NU
7	NU	NU	0	0	0	0	NU	NU	NU	NU	NU	NU
8	0	1	1	0	1	1	0	0	0	0	0	0
9	17	0	0	0	20	20	0	0	0	NU	0	0
10	57	50	57	44	0	44	NU	NU	NU	NU	NU	NU
11	0	0	0	0	0	0	0	0	0	0	1	0
12	50	60	67	86	67	100	0	0	0	NU	25	NU
13	17	0	0	0	0	10	NU	NU	NU	NU	0	NU
14	2	2	1	3	2	2	1	1	1	1	0	2
15	21	27	24	26	25	22	50	0	20	0	100	67
16	14	14	15	24	11	13	0	50	0	NU	40	33
17	19	19	19	16	19	17	13	11	20	24	16	16
18	NU	NU	50	NU	NU	50	48	43	37	35	42	45
19	14	14	16	18	14	19	67	65	65	55	69	72
20	16	17	15	14	15	19	17	16	17	15	14	12
OVERALL	3	3	3	3	3	3	2	2	3	2	3	2

because of CPU limits. A distinct difference is also shown between groups one and two in the percentage of residencies ended because of reaching a main memory limit, with many fewer terminations taking place because of this in group one (29% - 32% v 49% - 52%). This indicates that the effect of the category scheduling in group two was having a very significant impact with a larger number of processes entering main memory in a category with too small a main memory limit and having to be rescheduled, removed and brought in again before completing their work. It also shows that the standard category scheduling scheme (employed in group one), employing a set of four transitions associated with each category, obtains a better fit between process requirements and the categories they use, than the modified scheme (group 2) employing only three transitions.

The consistency of behaviour in the categories between runs is further demonstrated in Tables 4.8 and 4.9 showing the mean number of pages brought into main memory and the mean CPU time obtained per residency respectively. Again these figures indicate that the classification mechanism into categories is functioning in as much as those categories with larger resource allowances do in fact use more of those resources than categories which

Table 4.8

MEAN PAGE-IN REQUESTS PER RESIDENCY IN MAIN MEMORY

	EXPERIMENT RUN											
CATEGORY	A	C	E	G	I	K	B	D	F	H	J	L
1	49	49	48	50	49	48	50	50	47	51	50	38
2	18	19	19	19	19	19	17	18	17	17	18	17
3	30	30	28	29	29	30	24	24	23	24	23	23
4	35	33	35	37	34	35	26	26	27	25	29	27
5	15	15	16	16	15	15	17	17	17	17	17	17
6	NU	NU	20	NU	NU	NU	NU	NU	NU	NU	NU	NU
7	NU	NU	20	20	20	20	NU	NU	NU	NU	NU	NU
8	26	27	26	26	27	27	23	22	23	23	22	23
9	27	28	29	27	27	26	27	27	27	NU	19	27
10	31	31	34	32	35	31	NU	NU	NU	NU	NU	NU
11	37	37	38	37	37	38	27	27	27	27	27	26
12	37	39	42	32	35	39	32	40	28	NU	36	NU
13	43	46	38	43	44	44	NU	NU	NU	NU	27	NU
14	46	45	45	45	45	45	29	27	29	28	29	28
15	54	54	53	55	54	52	29	36	40	36	31	35
16	63	62	54	57	61	65	36	41	41	NU	48	51
17	60	59	59	60	61	61	41	38	40	35	37	37
18	NU	NU	60	NU	NU	64	61	53	52	55	57	61
19	63	63	66	65	67	65	51	47	52	50	51	52
20	39	41	39	42	38	44	41	40	38	41	38	38
OVERALL	35	35	35	36	35	35	26	25	25	25	25	25

Table 4.9

CPU TIME PER MAIN MEMORY RESIDENCY (all CPU times are uncorrected and quoted in milliseconds)

CATEGORY	A	C	E	G	I	K	B	D	F	H	J	L
1	158	145	147	152	144	142	146	146	136	146	150	111
2	52	41	37	37	40	39	26	28	28	28	32	31
3	41	39	42	42	39	39	45	42	44	49	46	45
4	77	57	72	88	68	73	76	66	74	66	92	73
5	17	26	31	27	30	28	22	24	26	21	26	26
6	N.U.	N.U.	1432	N.U.	N.U.	N.U.	N.U.	N.U.	N.U.	N.U.	N.U.	N.U.
7	N.U.	N.U.	1933	1202	1267	1042	N.U.	N.U.	N.U.	N.U.	N.U.	N.U.
8	60	59	64	62	64	64	53	53	56	51	55	57
9	3617	555	1140	609	2087	2148	721	651	629	N.U.	624	429
10	4328	3875	4315	3625	1541	4041	N.U.	N.U.	N.U.	N.U.	N.U.	N.U.
11	81	75	83	76	79	78	69	71	74	68	79	63
12	6120	8821	7467	9055	9247	10663	1362	4439	1671	N.U.	6905	N.U.
13	4061	2862	2959	1358	2480	2937	N.U.	N.U.	N.U.	N.U.	234	N.U.
14	104	102	103	109	106	99	80	74	97	77	81	99
15	5241	5561	5607	5629	5309	4852	7783	4021	5010	6436	10499	7187
16	4429	4122	3264	4996	5112	4809	2188	6413	2593	N.U.	7141	7887
17	691	700	756	666	728	777	653	584	833	773	729	697
18	N.U.	N.U.	5275	N.U.	N.U.	5680	5184	5018	4083	4589	4471	4937
19	1739	1594	1568	1878	1326	1522	5024	4218	4344	4114	4253	4269
20	494	610	572	592	583	613	616	725	611	588	564	449
OVERALL	281	304	289	341	302	322	231	260	242	293	271	317

are allocated less. Further information on the activity of the processes is given in Tables 4.10 and 4.11 showing the mean user CPU time between page faults, and the fraction of pages brought into main memory by the working set replacement policy. This shows, as would be expected, that group one processes do much more computing between page faults than group two and, that in those runs using the working set replacement policy, those pages brought in by prepaging will outnumber pages which are page faulted in, by a ratio of more than two to one.

Some other characteristics of the load placed on the system are summarised in Table 4.12. This clearly shows that the level of page-in requests involving transfers from tertiary memory is very low (approximately 2% - all preloading transfers come from the secondary memory). In fact of those transfers involved in the actual running of user processes (i.e. swapping in and out of main memory and page faulting) there is a ratio of 50:1 in favour of transfers involving secondary memory against those involving tertiary memory. The level of sharing in main memory is, as expected, seen to be dependent upon the amount of main memory available (and hence the level of multiprogramming). In the group one runs the requests for pages which are already in main memory (i.e. shared

Table 4.10

MEAN CPU TIME BETWEEN PAGE FAULTS (all CPU times are uncorrected and quoted in milliseconds)

CATEGORY	A	C	E	G	I	K	B	D	F	H	J	L
1	3	2	3	3	3	2	2	2	2	2	3	2
2	38	12	12	9	12	13	1	1	1	1	1	1
3	5	4	5	5	4	4	1	1	1	2	2	2
4	8	6	6	8	7	7	2	2	2	2	3	2
5	4	6	6	6	6	6	1	1	1	1	1	1
6	N.U.	N.U.	477	N.U.	N.U.	N.U.	N.U.	N.U.	N.U.	N.U.	N.U.	N.U.
7	N.U.	N.U.	644	400	422	347	N.U.	N.U.	N.U.	N.U.	N.U.	N.U.
8	17	15	18	17	16	16	2	2	2	2	2	2
9	657	92	157	91	347	370	27	24	24	N.U.	34	16
10	504	376	299	362	102	343	N.U.	N.U.	N.U.	N.U.	N.U.	N.U.
11	6	5	6	5	5	5	2	2	2	2	3	2
12	408	612	640	960	660	947	43	113	61	N.U.	195	N.U.
13	259	125	182	77	128	161	N.U.	N.U.	N.U.	N.U.	9	N.U.
14	7	7	7	7	7	6	2	2	3	2	2	3
15	298	323	344	352	337	347	270	113	127	183	349	211
16	229	199	247	287	256	218	61	159	64	N.U.	150	157
17	27	30	34	27	31	33	13	13	19	18	17	16
18	N.U.	N.U.	351	N.U.	N.U.	315	86	95	76	84	79	82
19	67	63	60	73	49	58	92	90	81	83	83	83
20	15	17	17	18	19	16	14	17	15	14	14	10
OVERALL	26	25	24	27	25	24	7	8	8	7	8	8

Table 4.11

PERCENTAGE OF ALL PAGE-IN REQUESTS WHICH ARE PRELOADS

	EXPERIMENT RUN											
CATEGORY	A	C	E	G	I	K	B	D	F	H	J	L
1	2	2	2	2	2	2	2	2	2	1	2	2
2	92	83	84	80	83	84	5	5	5	5	5	5
3	73	72	72	72	71	71	4	4	4	4	4	4
4	74	72	70	71	73	72	3	3	3	3	3	3
5	75	75	71	72	70	73	5	5	5	5	5	5
6	NU	NU	85	NU	NU	NU	NU	NU	NU	NU	NU	NU
7	NU	NU	85	85	85	85	NU	NU	NU	NU	NU	NU
8	87	86	86	86	86	85	4	4	4	4	4	4
9	79	78	75	75	78	78	3	3	3	NU	5	3
10	72	67	63	68	57	63	NU	NU	NU	NU	NU	NU
11	65	64	63	63	64	63	3	3	3	3	3	3
12	59	63	72	70	66	71	3	2	3	NU	2	NU
13	63	51	58	59	56	58	NU	NU	NU	NU	3	NU
14	69	68	68	67	68	68	3	3	3	3	3	3
15	67	68	69	71	71	73	3	2	2	2	3	2
16	69	66	75	69	67	66	2	2	2	NU	2	1
17	54	60	62	59	62	62	2	2	2	2	2	2
18	NU	NU	75	NU	NU	72	1	1	1	1	1	1
19	60	61	61	62	61	60	1	2	1	1	1	1
20	17	17	18	24	22	19	2	2	2	2	2	2
OVERALL	70	69	69	69	70	69	3	3	3	3	3	4

Table 4.12

FURTHER PAGING CHARACTERISTICS

(all figures are presented as percentages of all page-in requests)

EXP RUN	PRELOADING TRANSFERS	PAGE FAULTS: OVERALL	PAGE FAULTS: PAGE FOUND IN MAIN MEMORY	PAGE FAULTS: PAGE FOUND IN SECONDARY MEMORY	PAGE FAULTS: PAGE FOUND IN TERTIARY MEMORY	WRITES TO SECONDARY MEMORY	SHARED (I.E. NO TRANSFER)
A	48	30	10	18	2	33	33
C	45	30	12	17	1	33	36
E	49	30	10	19	2	33	30
G	48	31	10	19	2	33	32
I	53	30	8	20	1	33	26
K	52	30	9	20	1	33	27
B	4	96	32	62	2	36	32
D	4	96	34	61	2	37	34
F	4	96	30	64	2	37	30
H	4	96	30	64	2	37	30
J	4	96	26	69	2	37	26
L	4	96	26	69	2	37	26

pages) is evenly distributed between preloading and demand paging requests. The slight bias towards a higher level of sharing in demand paging requests in those runs is accounted for by new file page creations (two percent of all page-in requests) which appear as page faults for a page which is already in main memory (i.e. a shared page).

The level of loading on the secondary memory is shown in Table 4.13. These figures, obtained from the Q-sampling monitor, reveal that there was always a substantial amount of secondary memory unallocated and available (one and a half M-bytes of real drum space). Hence no use was made of the pseudo-drum space during these runs.

The influence of working set recalculations within any residency is shown in Table 4.14. This shows that very few pages are actually removed by this mechanism and that a large number of working set recalculations in fact remove no pages at all i.e. have no influence on the working set size. The higher percentage of pages removed by this mechanism seen in group one is caused by unwanted pages being preloaded. This preloading wastage i.e. pages which are preloaded but subsequently never used, ran consistently at 25% of all pages

Table 4.13

UNALLOCATED SECONDARY MEMORY DURING EXPERIMENT RUNS

EXP RUN	MEAN	MINIMUM
A	980	835
C	954	798
E	966	817
G	951	822
I	937	813
K	874	684
B	946	822
D	937	764
F	N O T A V A I L A B L E	
H	954	765
J	930	790
L	947	793

All figures presented in terms of unallocated secondary memory pages. Any use made of the pseudo drum would be indicated by an item dropping below 500.

Table 4.14

STROBING AND PARTIAL PRELOAD BEHAVIOUR

EXP RUN	NORMAL STROBES: RATIO STROBES : RESIDENCIES *	NORMAL STROBES: % OF STROBES NOT REMOVING ANY PAGES	EXTRA-STROBES: % OF RESIDENCIES CONTAINING AN EXTRA-STROBE	EXTRA-STROBES: % OF EXTRA-STROBES NOT REMOVING ANY PAGES	% OF PAGE-IN REQUESTS REMOVED BY STROBING	PROCESS LOADING: % RESIDENCIES STARTING WITH PARTIAL PRELOADS	PROCESS LOADING: % OF PARTIAL PRELOADS SUSPENDED
A	22 : 100	40	25	5	9	29	20
C	23 : 100	39	27	5	10	26	16
E	23 : 100	39	25	5	9	26	26
G	26 : 100	39	26	5	9	24	26
I	23 : 100	40	26	4	9	22	45
K	22 : 100	39	26	4	10	22	42
B	22 : 100	49	4	100	1	71	27
D	22 : 100	50	5	100	1	71	22
F	23 : 100	54	5	100	1	75	28
H	26 : 100	51	5	100	1	73	25
J	23 : 100	52	4	100	1	73	36
L	21 : 100	51	4	100	1	72	33

* This is not necessarily the same as the percentage of residencies containing a normal strobe as it is likely that certain residencies will contain several strobes.

which are preloaded. The advantages of the EXTRA-STROBE mechanism are shown in the group one runs where 95% of all EXTRA-STROBES do remove pages (unwanted preload pages). As would be expected, none of the EXTRA-STROBES in the group two runs remove any pages. The percentage of residencies which started with a partial preload is also shown, as is the percentage of those partial preloads which are subsequently suspended because the process tried to use more than the partial main memory allowance. The percentage of suspensions shows a tendency to increase as the main memory available decreases.

Conclusion

It is hoped that the reproducibility of the standard workload at the level of the resident supervisor has been shown and that the workload existing during the experiment adequately quantified. The processes involved may be noted to have quite large working sets (mean of 25 pages) and use little CPU time during any residency. Studies of the level of sharing on the main memory of EMAS have shown that virtually no sharing of user programmes or data takes place at this level, but that all sharing comes from DIRECTOR and subsystem code and common tables (file indices). Both DIRECTOR and subsystem will also

require further space for private data and working variables. This indicates that a high proportion of the contents of the working sets is made up of pages which are essentially system components so this large working set size is to a certain extent a consequence of the system structure

The difference in the category scheduling between the two different algorithms would also appear to have a distinct effect on the number of residencies required by a process in carrying out any piece of work. The algorithm, not employing working set replacement, requires a greater number of residencies per interaction, so the differences in performance observed between the two algorithms is not due solely to the paging-in mechanism.

Chapter 5

In this chapter the performance measurements obtained during the performance experiment are examined. The system is judged in terms of CPU utilisations, response times and throughput rates, with the major contributing factors to each of these being identified. The effect upon paging delay times of using a working set replacement policy is also investigated.

CPU Time Utilisation

The CPU time spent in each of the major states during the experiment runs is presented in Table 5.1. Rather than attempt adjustment to the $3 \times 2 \times 2^2$ factorial analysis to compensate for the loss of data incurred by the system crash at the end of run 'F', this data is considered as though from a $2 \times 2 \times 2$ factorial experiment (ignoring runs E, F, G and H - i.e. those with main memory at a level of 3/4 M-bytes). This data is analysed using the standard analysis of variance technique (Anova) for such experiments (cf. Appendix) [Cochran and Cox 1975, Johnstone and Leone 1964] calculated using Yates Algorithm [Yates 1937]. This identifies the effect each of the main factors has upon the system performance and this

Table 5.1

PERCENTAGE CPU TIME SPENT IN EACH OF THE MAJOR STATES

RUN	USER	SUPERVISOR	IDLE
A	54·3	42·7	3·0
B	51	46·3	2·7
C	52·9	38·3	8·8
D	46·4	39·5	14·1
E	52·3	41·7	5·9
F	N O T A V A I L A B L E		
G	51·9	35·8	12·3
H	41·7	37·7	20·6
I	49·3	39·0	11·7
J	41·8	40·5	17·6
K	44·1	33·7	22·3
L	37·5	34·3	28·3

effect is quantified for each in terms of the expected change in the performance caused when that factor is present at level two compared with the performance when the factor was at level one. As no replication of experiment runs took place an estimate must be obtained of the experimental error present in the results. This estimate is based upon the effect attributed to the higher order factors [Mendenhall 1968, Johnstone and Leone 1964], these higher order effects represent the interactions between major factors. The ratio of the mean squares of each factor and the error estimate is used in a simple F-test [Johnstone and Leone 1964] to test the significance of the average effect due to the major factors upon the overall system performance.

The percentage of the CPU time obtained by user processes is considered in Table 5.2. This shows the greatest contributing factor to be the change in the level of main memory - a change of 1/4 M-byte of memory causing eight percent more time to be spent in user state - followed by the software algorithm - a change of six percent - and the least influence to have been caused by the number of drum channels - a difference of four percent. The size of main memory is also the major contributing factor in reducing the amount of time absorbed by the idle state (Table 5.3) - nearly 13% of CPU time being added to the idle time by

Table 5.2

ANOVA Table for the Percentage of Time Spent in User State

(Mean 47·16)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MAIN MEMORY	-8·00	127·84	1	127·84	78·64 ****
DRUM CHANNELS	-3·89	30·19	1	30·19	18·57 **
SOFTWARE ALGORITHM	-5·97	71·16	1	71·16	43·78 ***
2nd ORDER EFFECTS					
MEMORY X CHANNEL	-0·91	1·64	1		
MEMORY X ALGORITHM	-1·04	2·14	1		
CHANNEL X ALGORITHM	-0·58	0·66	1	4	1·63
3rd ORDER EFFECT					
MEMORY X CHANNEL X ALGORITHM	1·02	2·06	1		
TOTAL		235·68	7		

FACTOR	LEVEL 1	LEVEL 2
MAIN MEMORY	7/8 M	5/8 M
DRUM CHANNELS	2	1
SOFTWARE ALGORITHM	WSR	PDP

** SIGNIFICANT AT 97·5% level (by F - test)

*** SIGNIFICANT AT 99 % level (by F - test)

**** SIGNIFICANT AT 99.9% level (by F - test)

Table 5.3

ANOVA Table for the Percentage of Time Spent in the Idle State

(Mean 13·56)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MAIN MEMORY	-12·83	328·96	1	328·96	84·53 ****
DRUM CHANNELS	- 9·61	184·70	1	184·70	47·46 ***
SOFTWARE ALGORITHM	- 4·23	35·79	1	35·79	9·20 *
2nd ORDER EFFECTS					
MEMORY X CHANNELS	0·99	1·94	1		
MEMORY X ALGORITHM	1·71	5·81	1		
CHANNELS X ALGORITHM	1·41	3·98	1	4	15·57
3rd ORDER EFFECT					
MEMORY X CHANNEL X ALGORITHM	- 1·39	3·84	1		
TOTAL		565·02	7		

FACTOR	LEVEL 1	LEVEL 2
MAIN MEMORY	7/8 M	5/8 M
DRUM CHANNELS	2	1
SOFTWARE ALGORITHM	WSR	PDP

**** SIGNIFICANT AT 99.9% level (by F - test)

*** SIGNIFICANT AT 99 % level (by F - test)

* SIGNIFICANT AT 95 % level (by F - test)

removing 1/4 M-bytes of main memory, whilst the removal of one of the two drum channels would account for 9·6% more of the total CPU time being spent in the idle state, and changing the software algorithm would result in just over four percent of the total CPU time being wasted in idle time.

One of the very striking features of the CPU utilisation data is the very large amount of CPU time being taken up by the system itself. EMAS would not appear to be unique in this respect [Sekino 1972, Bard 1971], in fact its supervisor overhead would appear to be lower than most systems of this class [Lynch 1975], though results on overhead of this nature are, perhaps understandably, not given great publicity. Of the factors covered by the experiment the removal of one of the two drum channels caused a drop in overhead of 5·7% (Table 5.4) - slightly more than the drop in user state CPU of just under four percent. Moving down a level in main memory causes a drop of just under five percent of the total time being spent in supervisor time, with a change in user state of eight percent. Meanwhile, using software algorithm at level two (PDP) causes 1·7% more of the time to go into the supervisor with a loss of user state of just under six percent, so the algorithm with WSR gives less time in supervisor and

Table 5.4

ANOVA Table for the Percentage of Time Spent in the Supervisor State

(Mean 39·27)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MAIN MEMORY	-4·85	47·00	1	47·00	71·31 ***
DRUM CHANNELS	-5·70	64·92	1	64·92	98·51 ****
SOFTWARE ALGORITHM	1·71	5·87	1	5·87	8·90 *
2nd ORDER EFFECTS					
MEMORY X CHANNELS	0·10	0·02	1		
MEMORY X ALGORITHM	-0·65	0·84	1		4
CHANNEL X ALGORITHM	-0·86	1·47			
3rd ORDER EFFECT					
MEMORY X CHANNEL X ALGORITHM	0·39	0·31	1		
TOTAL		120·41	7		

FACTOR	LEVEL 1	LEVEL 2
MAIN MEMORY	7/8 M	3/4 M
DRUM CHANNELS	2	1
SOFTWARE ALGORITHM		

**** SIGNIFICANT AT 99.9% level (by F - test)

*** SIGNIFICANT AT 99 % level (by F - test)

* SIGNIFICANT AT 95 % level (by F - test)

more time in user processes than the PDP algorithm.

The major supervisor functions which absorb this large amount of CPU time are shown in Table 5.5. It can easily be seen that the major contributor to the supervisor time is the organising of drum transfers i.e. the queueing of requests in the sector queues; removing requests from these queues and constructing channel command chains; fielding interupts at the completion of chains or after the completion of a demand page read (a programme controlled interrupt - PCI) and the sending of replies to the appropriate supervisor processes. Splitting this time into two major components - fielding requests and fielding interupts (Table 5.6) it may be seen that the size of memory has very little effect, and that the effect of going from two drum channels to one is to lower the amount of time spent in fielding requests by about 3·5%, but to increase the time spent fielding interupts by about two percent. This is probably due to the fact that in the two channel version channel chains are started more often on the arrival of requests (i.e. the requests find a channel free), whilst in the one channel case the lengths of channel chains are longer (Table 5.7) and channel chains are more likely to be started at the end of a previous channel chain i.e. when a channel chain completes it is more

Table 5.5

Percentage of Supervisor CPU Time Taken up by Major Functions

EXP RUN	DRUM TRANSFERS	DISC TRANSFERS	CORE LOADING	DRUM LOADING	CONTEXT SWITCHING	PROCESS CONTROL	SVC PARAMETER PASSING	COMMUNI-CATIONS	DEVICE POLLING	MAG TAPES
A	39·1	6·8	33·0	3·1	6·7	1·6	3·7	4·4	0·1	2·7
C	37·6	6·9	33·1	3·3	6·6	1·7	3·9	4·8	0·1	2·8
E	39·4	6·4	32·4	3·1	6·7	1·7	3·8	4·5	0·1	2·5
G	37·8	7·1	32·6	3·3	6·8	1·6	3·7	4·6	0·1	2·8
I	39·5	6·1	32·4	3·0	6·7	1·5	3·7	4·6	0·3	2·5
K	39·0	6·2	32·2	3·0	6·8	1·6	3·6	4·9	0·3	2·5
B	42·1	5·4	33·6	2·4	7·2	1·3	2·9	3·6	0·1	2·2
D	40·0	5·5	34·9	2·6	7·1	1·4	3·0	3·8	0·1	2·2
F	N O T	A V A I L A B L E								
H	40·4	5·2	34·5	2·4	7·3	1·3	3·0	3·7	0·3	2·0
J	42·9	5·0	33·6	2·4	6·8	1·2	2·8	3·4	0·3	2·0
L	41·2	4·7	34·4	2·2	7·3	1·2	2·8	3·7	0·3	1·7

Table 5.6

Percentage of Supervisor Time Absorbed by Drum Transfers

(A) ANOVA Table Fielding Transfer Requests Mean 21·94

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MAIN MEMORY	0·73	1·05	1	1·05	2·67
DRUM CHANNELS	-3·48	24·15	1	24·15	61·34 ***
SOFTWARE ALGORITHM	1·58	4·96	1	4·96	12·60 **
MEMORY X CHANNELS	0·08	0·01	1		
MEMORY X ALGORITHM	0·03	0·001	1		
CHANNELS X ALGORITHM	-0·88	1·53	1	4	0·46
MEMORY X CHANNELS X ALGORITHM	-0·13	0·03	1		
TOTAL		31·74	7		

(B) Fielding Channel Interupts Mean 18·16

MAIN MEMORY	0·28	0·15	1	0·15	2·37
DRUM CHANNELS	2·18	9·46	1	9·46	148·41 ****
SOFTWARE ALGORITHM	1·28	3·25	1	3·25	51·00 ***
MEMORY X CHANNELS	0·13	0·13	1		
MEMORY X ALGORITHM	0·03	0·001	1		
CHANNELS X ALGORITHM	0·33	0·21	1	4	0·14
MEMORY X CHANNEL X ALGORITHM	0·08	0·01	1		
TOTAL		13·12	7		

**** SIGNIFICANT AT 99·9% level
*** SIGNIFICANT AT 99 % level
** SIGNIFICANT AT 97·5% level

Table 5.7

Mean Length of Drum Channel Chains

EXP RUN	MEAN CHAIN LENGTH	INTERUPTS/SECOND	
A	2·8	47·1	
E	2·8	46·6	2 CHANNELS + WSR
I	2·8	43·5	
C	3·5	39·0	
G	3·2	37·6	1 CHANNEL + WSR
K	3·1	36·4	
B	1·6	62·7	
F	N O T A V A I L A B L E		2 CHANNELS + PDP
J	1·5	55·7	
D	2·0	46·6	
H	1·9	45·6	1 CHANNEL + PDP
L	1·7	43·6	

likely to find requests waiting in the sector queues than in the two channel case. The PDP algorithm consistently requires a higher percentage of supervisor time in both drum services caused by shorter channel claims and more interupts (as there is an interupt at the end of each demand page read). Indeed, if the throughput achieved on the drum is considered (Table 5.8), it can be seen that the WSR software algorithm has an average transfer rate of some ten pages per second more than the PDP algorithm for a lower overhead. Having more drum channels or main memory also show increases in throughput of around 9 and 8 pages per second respectively.

The other supervisor function which absorbs a considerable amount of CPU time is that of main memory loading. The subfunctions involved in this are:-

(a) Electing processes to the multiprogramming set and organising preloading transfers.

(b) Handling page faults.

(c) Recalculating working sets, removing processes from memory and organising transfers out.

Table 5.8

ANOVA Table

Mean Drum Transfer Rate per Second

Mean = 68·9 transfers/second

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MAIN MEMORY	-8·20	134·48	1	134·48	136·87 ****
DRUM CHANNELS	-9·15	167·44	1	167·44	170·42 ****
SOFTWARE ALGORITHM	-9·95	198·00	1	198·00	201·52 ****
2-FACTOR INTERACTIONS					
MEMORY X CHANNELS	-0·55	0·61	1		
MEMORY X ALGORITHM	-1·15	2·65	1		
CHANNELS X ALGORITHM	-0·3	0·18	1	4	0·98
3-FACTOR INTERACTION					
MEMORY X CHANNEL X ALGORITHM	+0·5	0·5	1		
TOTAL		503·86			

FACTOR	LEVEL 1	LEVEL 2
MAIN MEMORY	7/8 M-bytes	5/8 M-bytes
DRUM CHANNELS	2	1
SOFTWARE ALGORITHM	WSR	PDP

**** SIGNIFICANT AT 99·9% level (F - test)

(d) Fielding replies from transfer requests.

The factor having the greatest effect on the first three subfunctions is the software algorithm. This shows that the WSR algorithm spends much less time in handling page faults (Table 5.9) but more time in subfunction (a) - caused by organising preloading transfers - and more time in subfunction (b) organising page removals from core (a consequence of loading more pages). The average effect due to the software algorithm is an order of magnitude greater than that due to the two other factors. The effect of main memory size is the next significant factor in the first three subfunctions, with subfunction (c) appearing to be the most sensitive to this factor. Indeed, subfunction (c) is the only one in which the effect of the number of channels has any significant effect. The percentage of supervisor time in handling replies to transfer requests does not show any significant variation (above 90%) with any of the major factors - it will, however, vary in terms of the percentage of total CPU time in just the same way as the mean transfer rate through the drum.

The other two functions absorbing a reasonable amount of the time (Table 5.10) are context switching and disc transfer organisation. Context

Table 5.9

Subfunctions of Main Memory Loading as a Percentage of Total Supervisor Time

			MAJOR FACTORS - AVERAGE EFFECTS		
SUB	FUNCTION	MEAN	MEMORY	CHANNELS	ALGORITHM
(a)	MPS ELECTION	2·41%	-0·18 +	-0·13	-3·48 ****
(b)	PAGE FAULTS	11·65%	-0·41 *	0·25	7·85 ****
(c)	PAGE REMOVAL	7·34%	-0·38 ****	0·18 **	-2·63 ****
(d)	TRANSFER REPLIES	12·00%	0·3	-0·15	-0·3

**** SIGNIFICANT AT 99·9% level

** SIGNIFICANT AT 97·5% level

* SIGNIFICANT AT 95 % level

+ SIGNIFICANT AT 90 % level

Table 5.10

Significant Factors in Other Supervisor Functions
(Judged as Percentage of Total Supervisor Time)

		MAJOR FACTORS - AVERAGE EFFECTS		
FUNCTION	MEAN	MEMORY	CHANNELS	ALGORITHM
Context Switching	6·9%	-0·1	-0·1	-0·4 *
Disc Transfers	5·83	-0·65 ****	-0·1	3·65 ****

switching overhead is higher under the PDP* algorithm, owing to the fact that a context switch will be necessary after each page fault, for a page which is not found in main memory, and there are many fewer page faults under the WSR scheme. The percentage of supervisor time involved in organising disc transfers under the WSR scheme is larger than under the PDP algorithm. This is probably due to the fact that processes are getting through more work under the WSR scheme and hence require more disc transfers, caused by accessing more files (which will be page faulted-in from disc) and pages which lie unused on drum being moved back to disc by the drum working set algorithms.

If one was to try to identify why time-shared, virtual memory systems did not live up to the original hopes of their constructors, that is, hopes of several hundred terminals simultaneously active, it was probably caused by the size of the working sets being much larger than anticipated - it had been originally conjectured that EMAS processes would have working sets of around eight pages [Whitfield 1972], whereas thirty two pages has become the norm. This results in lower multiprogramming levels and many more transfers per process residency than intended. Combined with the unsuitability of the IBM-360 type of channel

architecture to the type of transfer rates required, this resulted in the very high percentage of time the expensive CPU is required to spend in supervisor state, thus lowering its availability to user processes. Any designer producing a powerful central processor intended to be used in a time shared, virtual memory environment would be well advised to consider ways of distributing the supervisor functions to less powerful special purpose processors (perhaps with order codes enabling fast table searching, which constitutes so much of supervisor work), and leave the main processor free only to do context switches and execute user programmes.

The mean levels of multiprogramming observed on the various experiment runs are shown in Table 5.11A. Processes are considered to be in the multiprogramming set from when they are first given a main memory allocation (and begin to preload at least their master page) until all pages belonging to that process have been removed from main memory. As these results (and most of the subsequent results in this chapter) are derived from the event trace data, the data is analysed as though for a 3 x 2 x 2 factorial experiment, again using a standard ANOVA technique and an algorithm suggested by Yates [Yates 1937]. As would be expected, the greatest effect

Table 5.11

(A) Mean Multiprogramming Levels on the Experiment Runs

EXPERIMENT RUN	A	C	E	G	I	K	B	D	F	H	J	L
MMPL	4·7	5·4	3·9	4·4	3·1	3·7	4·8	5·5	4·1	4·6	3·4	3·7

(B) ANOVA Table for Mean Multiprogramming Level (Mean = 4·3)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MEMORY					
7/8 3/4	-0·85)				
3/4 5/8	-0·77)	1·75	2	0·87	504·05 ****
7/8 5/8	-1·62)				
CHANNELS	0·54	0·30	1	0·30	167·39 ****
ALGORITHM	0·12	0·01	1	0·01	8·09 **
HIGH ORDER FACTORS (ERROR ESTIMATE)		0·01	7	0·002	
TOTAL		2·06	11		

**** SIGNIFICANT AT 99·9% level (F - test)

** SIGNIFICANT AT 97·5% level (F - test)

upon multiprogramming level comes from the size of main memory, followed by the number of channels available - with a higher multiprogramming level being seen in the runs with a single channel. This is explained by the definition of when a process is in the multiprogramming set. The time it takes to remove a process will be longer in the one channel case (results given later). Whilst this removal is taking place, a further process may be added to the MPS using pages freed by the process being removed but not requiring transfer out (about 2/3 of the removed process' working set). The overlap of these two processes will be longer in the single channel case due to longer transfer times, and the mean multiprogramming level will show this increase. The PDP algorithm also shows a higher multiprogramming level than the WSR algorithm due to the higher proportion of smaller memory categories observed in that algorithm's workload.

Response Times

The performance of the system as observed by the user i.e. the response time, is now considered (Table 5.12A). The response time is defined as the time the process spends in the awake state and this is derived from the event trace data. The factor having the greatest influence upon the mean response

Table 5.12

(A) Mean Response Times (in seconds) Observed in Experiment Runs

EXPERIMENT RUN	A	C	E	G	I	K
RESPONSE TIME	6·40	8·60	9·11	10·06	10·75	13·82

EXPERIMENT RUN	B	D	F	H	J	L
RESPONSE TIME	11·74	17·5	18·63	18·65	22·01	24·16

(B) ANOVA Table for Response Times (Mean = 14·29)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MEMORY 7/8 3/4 3/4 5/8 7/8 5/8	3·05) 3·57) 6·63)	29·34	2	14·67	18·64 ***
CHANNELS	2·36	5·56	1	5·56	7·07 *
ALGORITHM	8·99	80·87	1	80·87	102·73 ****
HIGH ORDER FACTORS (ERROR ESTIMATE)		5·51	7	0·79	
TOTAL		121·28	11		

**** SIGNIFICANT AT 99·9% level (F - test)

*** SIGNIFICANT AT 99 % level (F - test)

* SIGNIFICANT AT 95 % level (F - test)

time is found to be the software algorithm with the WSR algorithm producing a response time on average some nine seconds less than the PDP algorithm. The next most influential factor is that of main memory size, with some three seconds being added to the response time when moving from 7/8 M-bytes to 3/4 M-bytes, and some 3·5 seconds when moving from 3/4 to 5/8 M-bytes. The lowest contribution to the response time seems to come from the number of drum channels, the removal of one of the two channels causing an increase in response time of just under 2·5 seconds.

The contributing factors to these response times are shown in Table 5.13 and 5.14 in terms of the expected wait time per entry to each significant station in the Process Management Model, and the expected number of times per interaction that a process would enter any of these stations. These figures are shown again in Table 5.15 normalised with respect to the CPU times. These stretch factors show the expected wait times encountered in obtaining a unit quantity of CPU time, and the major areas from which this wait time arises: the RUN queues (awaiting allocation of CPU), the CORE queues (awaiting allocation of main memory), paging into main memory (preloading and page faulting) and paging out of memory (writing back to the drum). As with the

Table 5.13

Expected Wait Time per Entry in Each Major Station (in milliseconds)

EXP RUN	ASQ	CQ1	CQ2	CQ3	CQ4	CQ5	PREL	SUSP	RQ1	RQ2	CPU	PFLT	PTAKE	ATQ
A	2	1405	3013	4326	6310	42788	202	0·18	21	105	16·0	38	192	3058
C	2	1238	3431	4619	5194	51745	381	0·13	20	82	17·3	58	437	3715
E	2	2156	4517	6621	9166	49625	212	0·16	18	63	15·6	36	199	3041
G	3	1704	3907	5495	7726	64665	364	0·16	15	56	15·9	49	399	3643
I	2	2578	6611	8101	9273	81625	218	0·16	12	29	15·6	35	198	2900
K	2	2960	8607	10934	13233	129409	366	0·00	11	24	16·1	47	388	2765
B	2	2355	3662	15060	6888	69848	65	0·00	13	42	5·7	27	118	2598
D	2	3687	5882	18031	9135	30608	122	0·15	13	32	6·5	44	214	2522
F	2	4481	6877	24840	6703	33855	66	0·18	11	26	6·1	30	115	2801
H	2	3566	5614	39119	11218	32187	113	0·18	10	21	6·1	42	200	2992
J	1	6057	8628	26136	7022	28442	67	0·16	9	13	6·4	27	112	2372
L	2	5851	9385	41715	9835	38197	114	0·18	6	11	6·2	39	184	2242

Table 5.14

Mean Number of Entries to Each Station per Interaction

EXP RUN	ASQ	CQ1	CQ2	CQ3	CQ4	CQ5	PREL	SUSP	RQ1	RQ2	CPU	PFLT	PTAKE	ATQ
A	0·09	0·80	0·51	0·11	0·03	0·04	1·5	0·09	12·3	8·4	20·7	15·1	1·5	0·09
C	0·09	0·78	0·54	0·11	0·03	0·04	1·5	0·07	12·5	7·0	19·5	16·1	1·5	0·09
E	0·09	0·82	0·49	0·11	0·03	0·04	1·5	0·11	13·2	7·8	21·0	15·8	1·5	0·09
G	0·10	0·82	0·51	0·13	0·03	0·05	1·5	0·10	13·6	8·1	21·7	16·7	1·5	0·11
I	0·09	0·81	0·51	0·11	0·03	0·04	1·5	0·15	13·7	7·1	20·8	15·6	1·5	0·10
K	0·09	0·82	0·51	0·11	0·03	0·04	1·5	0·14	13·7	6·3	20·0	15·7	1·5	0·09
B	0·11	1·32	0·61	0·16	0·03	0·02	2·1	0·4	37·3	17·1	54·4	52·4	2·1	0·11
D	0·11	1·32	0·62	0·19	0·03	0·03	2·1	0·4	36·8	13·9	50·7	52·2	2·1	0·11
F	0·10	1·37	0·59	0·17	0·03	0·03	2·1	0·5	38·5	14·7	53·2	52·6	2·1	0·11
H	0·10	1·36	0·63	0·19	0·03	0·03	2·1	0·4	39·7	12·6	52·3	53·9	2·1	0·11
J	0·10	1·34	0·57	0·18	0·03	0·03	2·1	0·6	39·9	12·5	52·4	51·5	2·1	0·10
L	0·09	1·37	0·54	0·18	0·03	0·02	2·1	0·5	39·3	10·5	49·8	50·6	2·1	0·10

Table 5.15

Stretch Factors

EXP RUN	OVERALL STRETCH FACTOR	DUE TO RUN QUEUES	DUE TO CORE QUEUES	DUE TO PAGING IN	DUE TO PAGING OUT
A	16·99	3·46	10·04	2·64	0·86
C	25·89	2·47	17·01	4·47	1·94
E	27·63	2·27	21·73	2·72	0·91
G	29·57	1·93	21·74	4·11	1·79
I	35·48	1·19	30·68	2·70	0·91
K	48·96	0·97	42·18	4·02	1·79
B	40·81	3·87	31·13	5·00	0·81
D	51·64	2·83	39·52	7·89	1·40
F	56·19	2·60	47·54	5·28	0·77
H	63·97	2·12	52·70	7·79	1·36
J	62·34	1·57	55·38	4·58	0·71
L	79·49	1·26	69·83	7·15	1·25

response time, the greatest influence upon the Stretch Factor (Table 5.16) is the algorithm used, followed by the size of main memory, and least influence caused by the number of drum channels.

The time spent in the Run Queues (Table 5.17) is influenced most by the size of main memory, as would be expected, the larger main memory sizes (with larger multiprogramming levels) cause a large proportion of wait time to be spent in the Run Queues. Next in order of influence is the number of channels available, with more time being spent in the Run Queues in the two channel case (despite the fact that the one channel case gives a higher level of multiprogramming) - due to the fact that those processes in the multiprogramming set are spending less of their time in the page wait state. The least influence is exerted by the algorithm, with the PDP algorithm delayed more in the Run Queues - probably caused by the higher level of multiprogramming seen in that case.

The component with the greatest influence, by almost an order of magnitude, upon the Stretch Factor is the time spent in the Core Queues (Table 5.18). This is most heavily influenced by the algorithm used, with the PDP algorithm spending much

Table 5.16

ANOVA Table - Stretch Factors (Mean = 44·91)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MEMORY					
7/8 3/4	+10·51)				
3/4 5/8	+12·23)	345·25	2	172·63	40·22 ****
7/8 5/8	+22·73)				
CHANNELS	10·01	100·27	1	100·27	23·36 ***
ALGORITHM	28·32	802·02	1	802·02	186·86 ****
HIGH ORDER FACTORS (ERROR ESTIMATE)		30·05	7	4·29	
TOTAL		1277·57	11		

Table 5.17

ANOVA Table - Run Queue Component of Stretch Factor (Mean = 2·21)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MEMORY					
7/8 3/4	-0·93)				
3/4 5/8	-0·98)	2·43	2	1·22	75·67 ****
7/8 5/8	-1·91)				
CHANNELS	-0·56	0·32	1	0·32	19·65 ***
ALGORITHM	0·33	0·11	1	0·11	6·72
HIGH ORDER FACTORS (ERROR ESTIMATE)		0·11	7	0·016	
TOTAL		2·97	11		

**** SIGNIFICANT AT 99·9% level (F - test)

*** SIGNIFICANT AT 99 % level (F - test)

* SIGNIFICANT AT 95 % level (F - test)

Table 5.18

ANOVA Table - Core Queue Component of Stretch Factor
(Mean = 36·62)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MEMORY					
7/8 3/4	11·50)				
3/4 5/8	13·59)	420·60	2	210·30	51·49 ***
7/8 5/8	25·09)				
CHANNELS	7·75	59·99	1	599·85	14·69 ***
ALGORITHM	25·45	647·80	1	647·80	158·61 ****
HIGH ORDER FACTORS (ERROR ESTIMATE)		28·59	7	4·08	
TOTAL		1156·97	11		

Table 5.19

ANOVA Table - To Main Memory Paging Component of Stretch Factor
(Mean = 4·86)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MEMORY					
7/8 3/4	-0·02)				
3/4 5/8	-0·36)	0·12	2	·06	1·00
7/8 5/8	-0·38)				
CHANNELS	2·08	4·34	1	4·34	70·23 ****
ALGORITHM	2·84	8·07	1	2·84	130·41 ****
HIGH ORDER FACTORS (ERROR ESTIMATE)		0·43	7	·06	
TOTAL		12·97	11		

**** SIGNIFICANT AT 99·9% level (F - test)

*** SIGNIFICANT AT 99 % level (F - test)

more time in the core queues - caused by the fact that the PDP algorithm requires an extra 0·5 residency, on average, per interaction which will result in an extra 0·5 entries to a core queue per interaction over that required by the WSR algorithm. Next most influential is the size of main memory, with small memory sizes spending more time in the Core Queues. Least influence is again exerted by the number of drum channels, with the single channel case spending more time in the Core Queues than the two channel one.

After the time spent in the Core Queues the next most influential component is the time spent in paging-in to core (Table 5.19). Again, the greatest influence on this component is the algorithm used, with the PDP algorithm (caused by its different paging-in discipline and the extra 0·5 residencies per interaction) spending much more time paging-in than the WSR algorithm. The number of channels available is next most influential, with less time being spent in paging-in when two channels are available. The size of main memory does not, however, appear to have any significant effect on the time spent on paging-in to main memory.

The least influential component of the Stretch Factor is the time spent paging out of

main memory (Table 5.20). Again, the size of main memory is found to have no significant effect, whilst the most influence seems to be exerted by the number of channels with greater delay times being incurred by single channel configurations. The effect of algorithm shows that less time is spent in paging out by the PDP algorithms than the WSR, despite the fact that more pages are written back per interaction.

Paging Behaviour

The mean effective page wait times in each type of paging is shown in Table 5.21. This shows the expected delay time incurred by a process from the time a request is issued on its behalf to have a page transferred, to when that page arrives in main memory, and the process is notified that the page is ready for it to use.

In the case of group one experiment runs (WSR algorithm) it can be observed that the mean delay time for the two classes of paging which involve bulk transfers (i.e. several transfer requests for a process being fired off at the same time) - preloading and write back - are both much shorter than page faults coming from the drum, the delay time for a preloading page transfer being about 1/3 of the demand page fault

Table 5.20

ANOVA Table - From Main Memory Paging Component of Stratch Factor

(Mean = 1·21)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MEMORY					
7/8 3/4	-0·05)				
3/4 5/8	-0·04)	0·005	2	0·003	0·43
7/8 5/8	-0·09)				
CHANNELS	0·76	0·10	1	0·10	100·38 ****
ALGORITHM	-0·32	0·58	1	0·58	17·59 ***
HIGH ORDER FACTORS (ERROR ESTIMATE)		0·04	7	0·006	
TOTAL		0·73	11		

**** SIGNIFICANT AT THE 99·9% level (F - test)

*** SIGNIFICANT AT THE 99 % level (F - test)

Table 5.21

Mean Delay Time per Page Transferred (in milliseconds)

EXP RUN	PRELOADING TRANSFERS	PAGE FAULT (ON DRUM)	PAGE FAULT (ON DISC)	PAGE FAULT (IN CORE)	WRITE TRANSFERS	ALL PAGE FAULTS
A	12	37	263	2	18	38
C	23	78	238	5	41	58
E	12	37	216	3	19	36
G	21	62	193	5	37	49
I	11	35	219	3	18	35
K	20	54	199	5	37	47
B	65	36	194	1	12	27
D	121	64	206	1	23	44
F	65	36	302	1	12	30
H	113	58	164	1	21	42
J	67	33	172	1	12	27
L	113	50	160	2	20	39

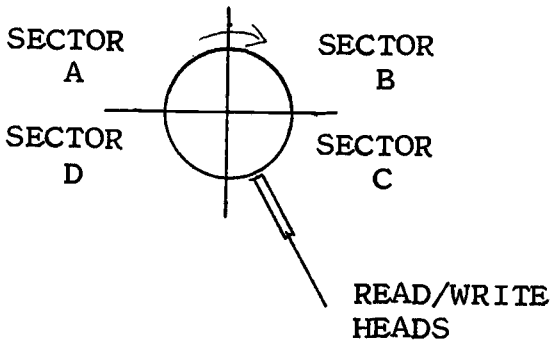
time, and write backs about 1/2 of the time for the demand page. This is despite the fact that drum transfer requests are ordered in the four sector queues in such a way that demand paging reads always have priority over prepaging reads, which in turn always have priority over writes (there is also a further priority ordering in terms of the physical drum which the request is destined for). Demand paging reads present in a channel chain containing other transfers after it will also cause a PCI to be generated when they complete, and thus the process will be notified of the page's arrival earlier than if it had to wait for the whole channel chain to complete (as is the case with prepaging and write back transfers). The effect of this priority scheme can be seen in the case of preloading transfers (for the master page) under the PDP algorithm. The delay time observed in the case of page faults for a page which is already in main memory is caused by pages which are owned by another process, and have a page frame allocated, but are still being transferred. Thus the page faulting process has to wait until this transfer completes.

This advantage of 'bulk transfers' over single transfers in terms of delay time per page is caused by the characteristics of the secondary

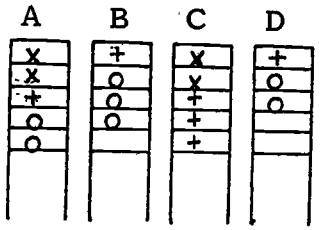
memory device - the drum, through which the overwhelming majority of the paging traffic passes. There is no position sensing on the drums used on the configurations measured. There will always, then, be a latency delay at the start of any drum channel chain with a mean of half a drum revolution (10 milliseconds) confirmed by measurements taken on these devices [Adams, Gelenbe and Vicard 1977]. After the latency the channel chain will be executed transferring a page each time there is a request in the chain corresponding to the drum sector under the read heads on the device - i.e. a request corresponding to the current sector window. There were up to eight sector windows covered by any drum channel chain during the experiment i.e. when a chain was being constructed the top request was removed from each sector queue in turn, twice (Figure 5.1). When a bulk transfer takes place the sector queues will be longer when the channel chain is constructed and there will be more transfers in each chain (Table 5.7) fewer sector windows will be 'lost' by having no transfer request corresponding to them and more transfers will take place for each latency delay.

Figure 5.1

OPERATION OF DRUM



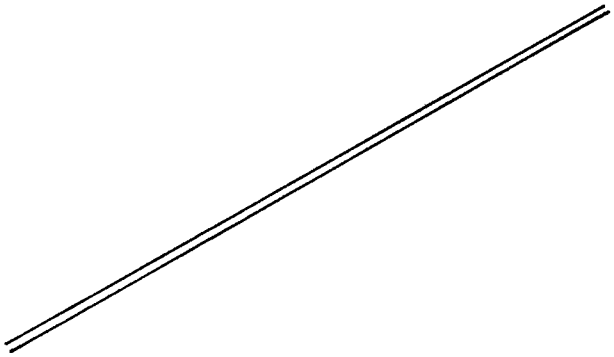
SECTOR QUEUES

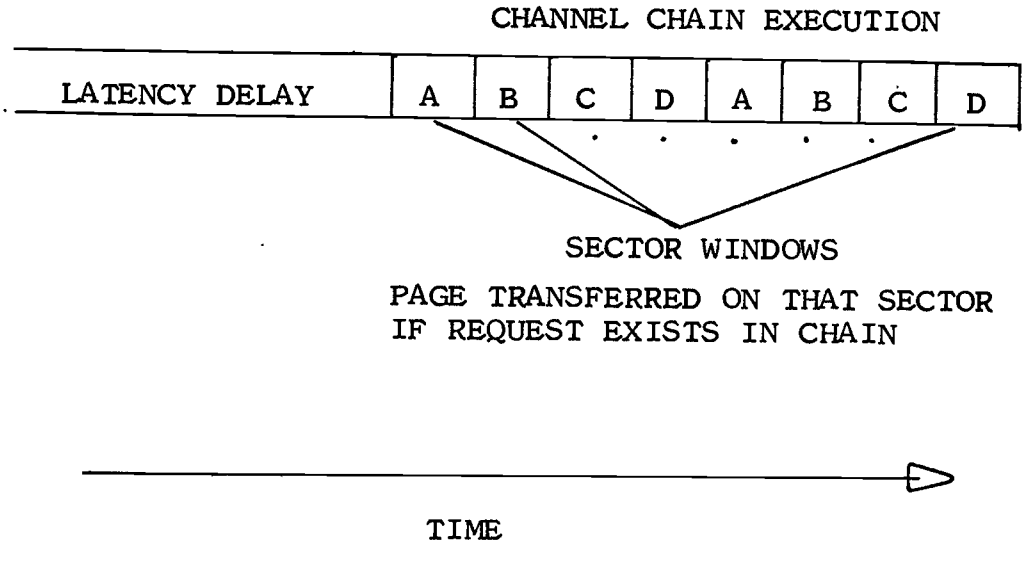


x demand page reads

\+ prepage reads

o write backs





TIME

Bulk Transfer Times

The ANOVA table for the mean effective page wait in transferring a preloaded page reveals that this delay time is not significantly influenced by the size of main memory (Table 5.22) but is very sensitive to the algorithm used - this is to be expected as preloading transfers do not, in fact, involve bulk transfers in the case of the PDP algorithms, where only a single page (the master page) is requested at a time. The number of channels also has an impact, the addition of a second channel causing a reduction of nearly 30 milliseconds per page. The other form of paging involving bulk transfers is that of page writes back to the drum when a process is being removed from main memory (an insignificant number of transfers are generated by strobing). Unlike preloading transfers writes back to drum take place as bulk transfers under both algorithms. The greatest impact upon this delay time is caused by the number of channels available with the removal of a channel adding an average of 14·5 milliseconds to the expected wait time per page (Table 5.23). The PDP algorithm is also found to have an expected delay time some 11·5 seconds lower than the WSR algorithm. The main cause of this is probably not so much the fact that fewer pages per residency are written back by this algorithm, but the fact that

Table 5.22

ANOVA Table for Mean Effective Page Wait Time (in milliseconds) per Preloaded Page Transfer (Mean = 53·6)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MEMORY					
7/8 3/4	-2·5)				
3/4 5/8	0)	5·6	2	2·8	0·05
7/8 3/4	-2·5)				
CHANNELS	29·8	890·0	1	890·0	14·9 ***
ALGORITHM	74·2	5500·7	1	5500·7	92·1 ****
HIGH ORDER FACTORS (ERROR ESTIMATE)		418·3	7	59·8	
TOTAL		6814·3	11		

Table 5.23

ANOVA Table for Mean Effective Page Wait Time (in milliseconds) per Write Transfer (Mean = 22·5)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MEMORY					
7/8 3/4	- 1·3)				
3/4 5/8	- 0·5)	2·2	2	1·1	0·2
7/8 3/4	- 1·8)				
CHANNELS	14·7	215·1	1	215·1	47·6 ****
ALGORITHM	-11·7	136·1	1	136·1	30·1 ****
HIGH ORDER FACTORS (ERROR ESTIMATE)		31·6	7	4·5	
TOTAL		385·0	11		

**** SIGNIFICANT AT THE 99·9% level (F - test)

*** SIGNIFICANT AT THE 99 % level (F - test)

there are not so many preloading transfers. When the system is working under the WSR algorithm and a process is removed from main memory, then the pages that this process owns, which do not require writing back, are released at once (2/3 of the working set) and the write-back requests issued for those which do. It is quite possible that the pages released immediately are adequate to allow another process to enter the MPS and a set of preload reads will be issued (within about 10 milliseconds of the set of write requests). These preload requests will arrive whilst some, if not all, of the write requests are still in the sector queues, and will take precedence over them, thus increasing the average write-back time. Under the PDP algorithm when a process is removed from main memory a similar thing will happen, and some pages will be released whilst a set of write requests will be fired off. Any process entering the MPS at that time will issue a single preloading request (for the master page) which will still take precedence over the writes. The interference caused to the leaving process, to the loading process will thus be much less. It may also be noted from Table 5.21 that the mean delay time for writes back under the PDP algorithm correspond very closely to the mean delay time for preloads under the WSR algorithm.

Page Faulting Delays

The delay time incurred in page faulting a page in from the drum is found to be sensitive to the size of main memory (Table 5.24) which may be explained by the fact that the larger memory sizes (which have longer delay times) also have higher drum transfer rates and as is seen later the drum page fault time is very sensitive to the paging rate. This page faulting time is, however, not significantly influenced by the algorithm (perhaps proving the worth of the priority given to demanded pages in the sector queues), but is greatly influenced by the number of channels available with a lengthening of the wait time of around 25 milliseconds being experienced when one of the two channels is removed. The time spent awaiting page faults which have to be transferred from the disc does not seem to be consistently influenced by any of the major factors included in the experiment in any significant way (Table 5.25). The small amount of time spent awaiting page faults to 'pages in main memory' is not significantly influenced by the size of main memory (Table 5.26) but is most sensitive to the algorithm used, with the PDP algorithm causing a delay of 2·8 milliseconds less than the WSR, due probably to the fact that under that algorithm the shared page which is being transferred-in is one of a set of

Table 5.24

ANOVA Table for Mean Effective Page Wait Time (in milliseconds) per Page, Page Faulted from Drum (Mean = 48·3)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MEMORY					
7/8 3/4	- 5·5)				
3/4 5/8	- 5·3)	77·1	2	38·5	4·0 +
7/8 5/8	-10·8)				
CHANNELS	25·3	641·8	1	641·8	66·8 ****
ALGORITHM	- 4·3	18·8	1	18·8	2·0
HIGH ORDER FACTORS (ERROR ESTIMATE)		67·3	7	9·6	
TOTAL		804·9	11		

Table 5.25

ANOVA Table for Mean Effective Page Wait Time (in milliseconds) per Page, Page Faulted from Disc (Mean = 210·5)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MEMORY					
7/8 3/4	- 6·5)				
3/4 5/8	-31·3)	1086·2	2	543·1	1·1
7/8 5/8	-37·8)				
CHANNELS	-34·3	1178·8	1	1178·8	2·311
ALGORITHM	-21·7	469·4	1	469·4	0·9
HIGH ORDER FACTORS (ERROR ESTIMATE)		3569·9	7	510·0	
TOTAL		6304·1	11		

**** SIGNIFICANT AT THE 99·9% level (F - test)

\+ SIGNIFICANT AT THE 90 % level (F - test)

Table 5.26

ANOVA Table for Mean Effective Page Wait Time (in milliseconds) per Page, Page Faulted in Core (Mean = 2·5)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MEMORY					
7/8 3/4	0·3)				
3/4 5/8	0·3)	0·2	2	0·1	0·5
7/8 5/8	0·5)				
CHANNELS	1·3	1·8	1	1·8	9·7 **
ALGORITHM	-2·7	7·1	1	7·1	39·0 ****
HIGH ORDER FACTORS (ERROR ESTIMATE)		1·3	7	0·2	
TOTAL		10·3	11		

Table 5.28

ANOVA Table for the Average Throughput - Interactions Completing per Second (Mean = 1·23)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MEMORY					
7/8 3/4	-0·14)				
3/4 5/8	-0·08)	0·034	2	0·017	49·5 ****
7/8 5/8	-0·22)				
CHANNELS	-0·152	0·023	1	0·023	67·2 ****
ALGORITHM	-0·335	0·112	1	0·112	327·7 ****
HIGH ORDER FACTORS (ERROR ESTIMATE)		0·002	7	0·0003	
TOTAL		0·17	11		

**** SIGNIFICANT AT THE 99·9% level (F - test)

** SIGNIFICANT AT THE 97·5% level (F - test)

preloads (twice as probable as the case when the shared page is another page faulted page). The arrival of the preloaded page will not be notified to its owners until the end of the channel chain containing it rather than immediately it arrives in main memory as is the case with demand page reads. The addition of a second channel also causes this delay time to be reduced by just over a millisecond as is to be expected as the two channel configurations consistently transfer pages quicker than the single channel case.

Influence of Paging Rate Upon Paging Delays

The major factor in the paging delays accrued by processes on EMAS is that of transfers involving the secondary memory - or drum. To observe how the drum paging characteristics vary with the drum paging rate, the event trace data in each of the experiment runs was partitioned into intervals of two seconds (of real time) and the numbers of page transfers in each type of drum paging as well as the mean delay time for each was calculated. Two seconds was chosen as being large enough to eliminate most end effects (being a factor of five larger than the largest mean delay time - total time in write-back) and yet small enough to still show differences between paging rates. This data is presented in terms of a set of graphs

(Figures 5.2 - 5.13) which show:-

(a) The relationship between expected effective page wait time per demand page read (from the drum) and the drum paging rate.

(b) The relationship between the average number of drum page faults and the drum paging rate.

(c) The relationship between the expected effective page wait time per page for preload reads and the drum paging rates.

(d) The relationship between the number of preloading transfers and the drum paging rate.

(e) The relationship between the expected effective page wait time for drum writes and the drum paging rate.

(f) The relationship between the number of write transfers and the drum paging rate.

(g) The frequency distribution of the paging rates.

It may be observed from these graphs that the effect of main memory is merely to increase the spread of paging rates with higher paging rates

Table 3.1

THE EXPERIMENTAL RUNS

EXPERIMENT NUMBER	PRIMARY MEMORY (M-BYTES)	CHANNELS TO SECONDARY MEMORY	SCHEDULING ALGORITHM
A	7/8	2	WSR
B	7/8	2	PDP
C	7/8	1	WSR
D	7/8	1	PDP
E	3/4	2	WSR
F	3/4	2	PDP
G	3/4	1	WSR
H	3/4	1	PDP
I	5/8	2	WSR
J	5/8	2	PDP
K	5/8	1	WSR
L	5/8	1	PDP

WSR - Using Working Set Replacement Policy

PDP - Using Pure Demand Paging scheme

All experiments were carried with a fixed workload of 32 simulated users. Hardware consisted of the ERCC ICL 4-75 (machine "B" complex) with 3 drums + 1 pseudo drum. EMAS, version 814, was used throughout, as were the executive processes - Volumes version 834, Demons version 877. Runs A-D used DIRECTOR 871, E-L used DIRECTOR 872 (a minor error corrected).

August 1975

Figure 5.2 (a)

EXPERIMENT A

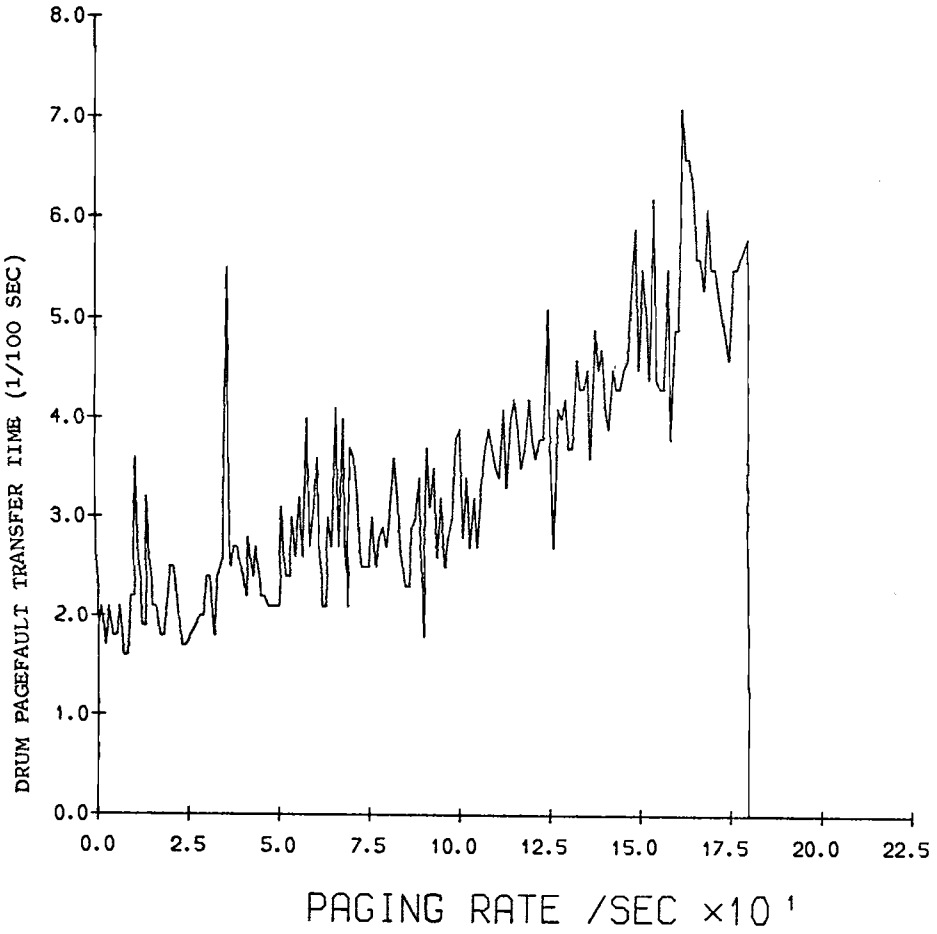


Figure 5.2 (b)

EXPERIMENT A

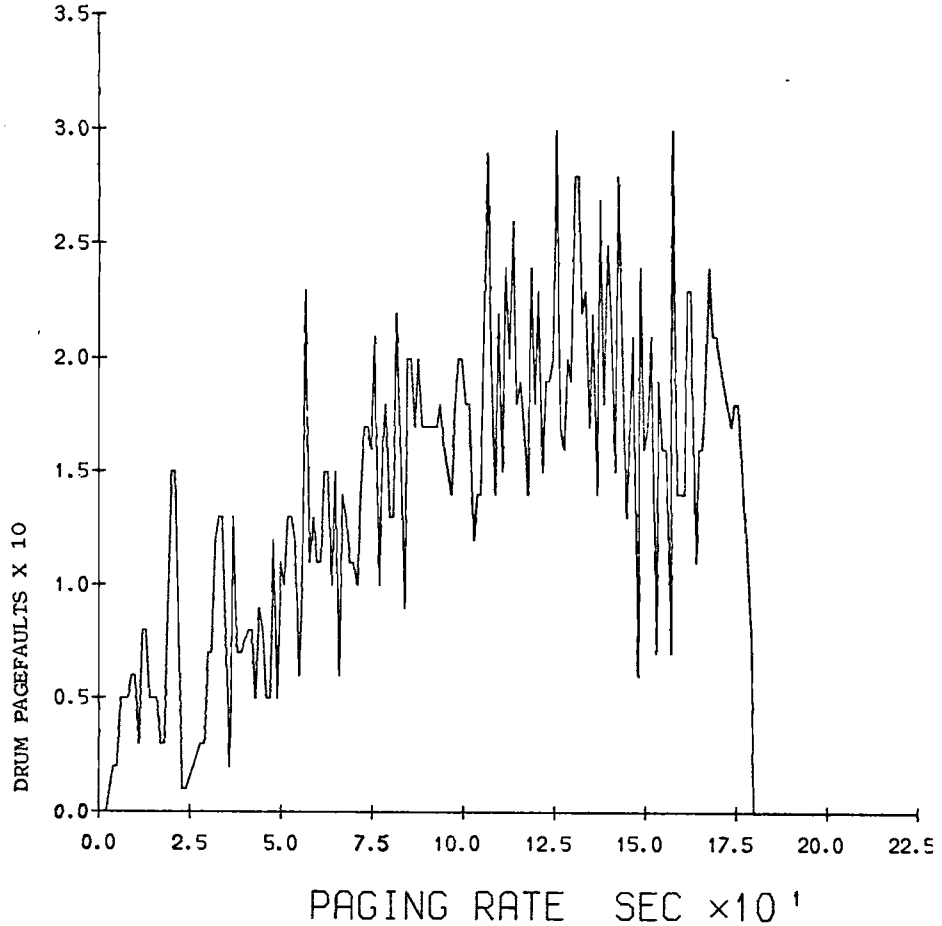


Figure 5.2 (c)

EXPERIMENT A

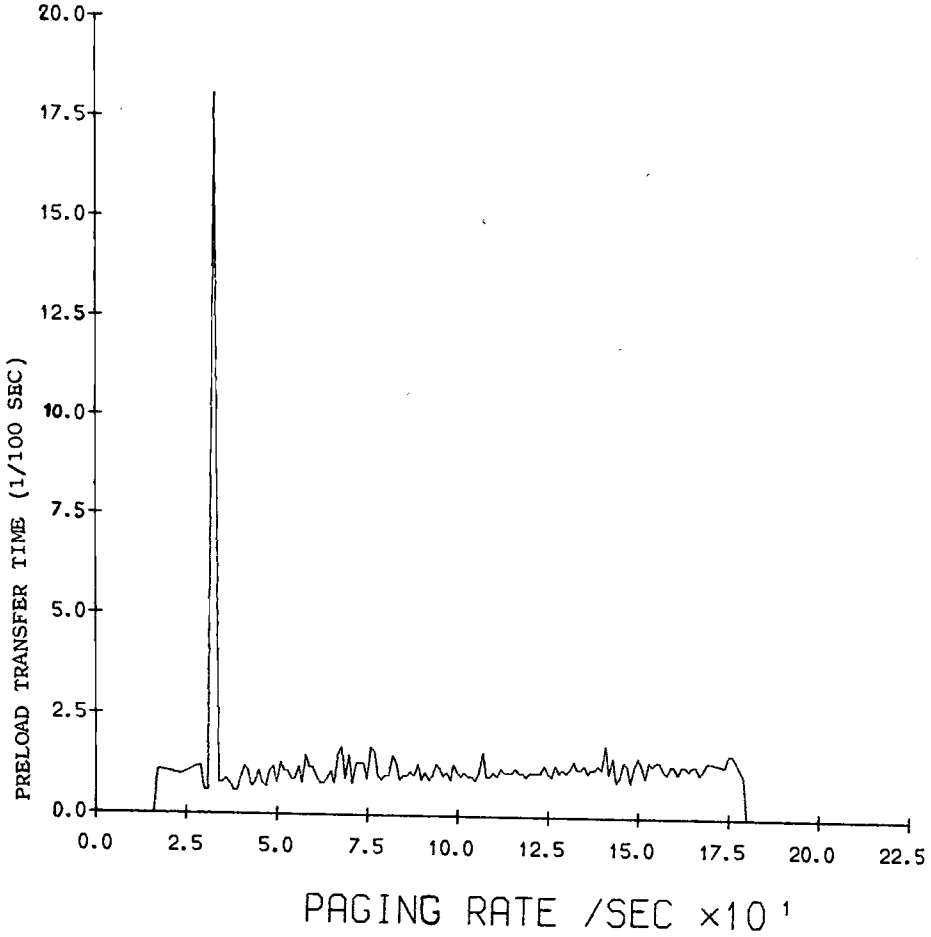


Figure 5.2 (d)

EXPERIMENT A

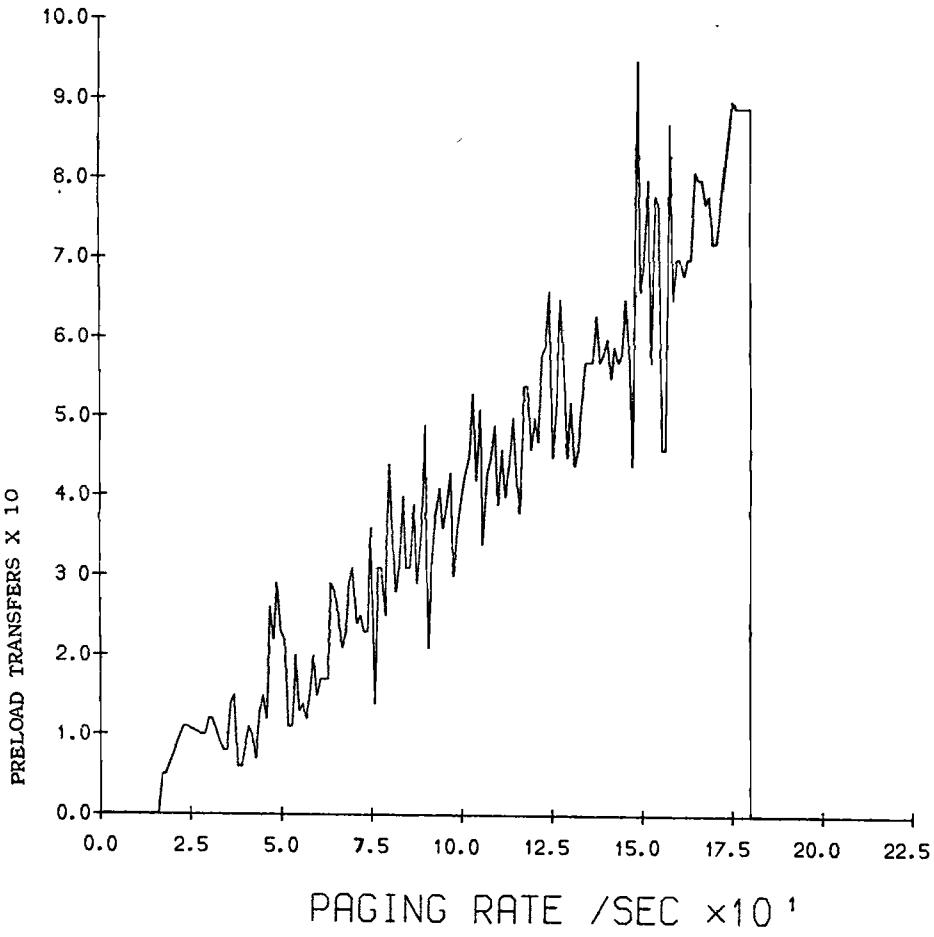


Figure 5.2 (e)

EXPERIMENT A

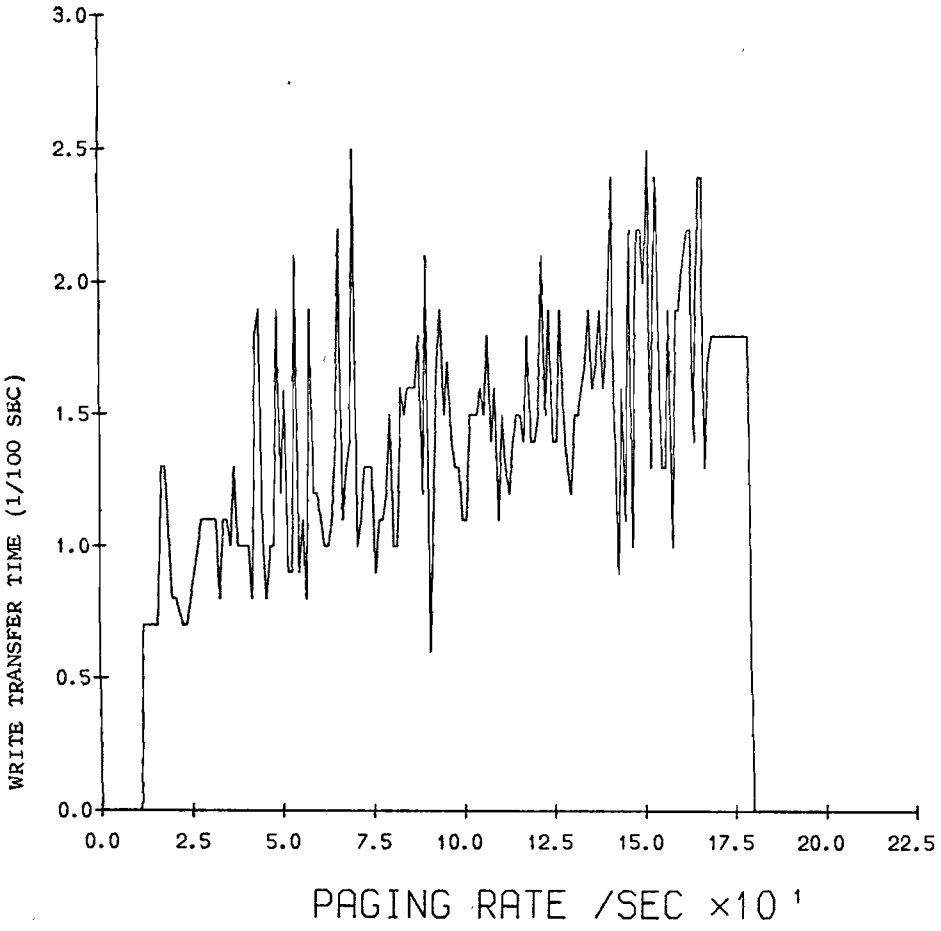


Figure 5.2 (f)

EXPERIMENT A

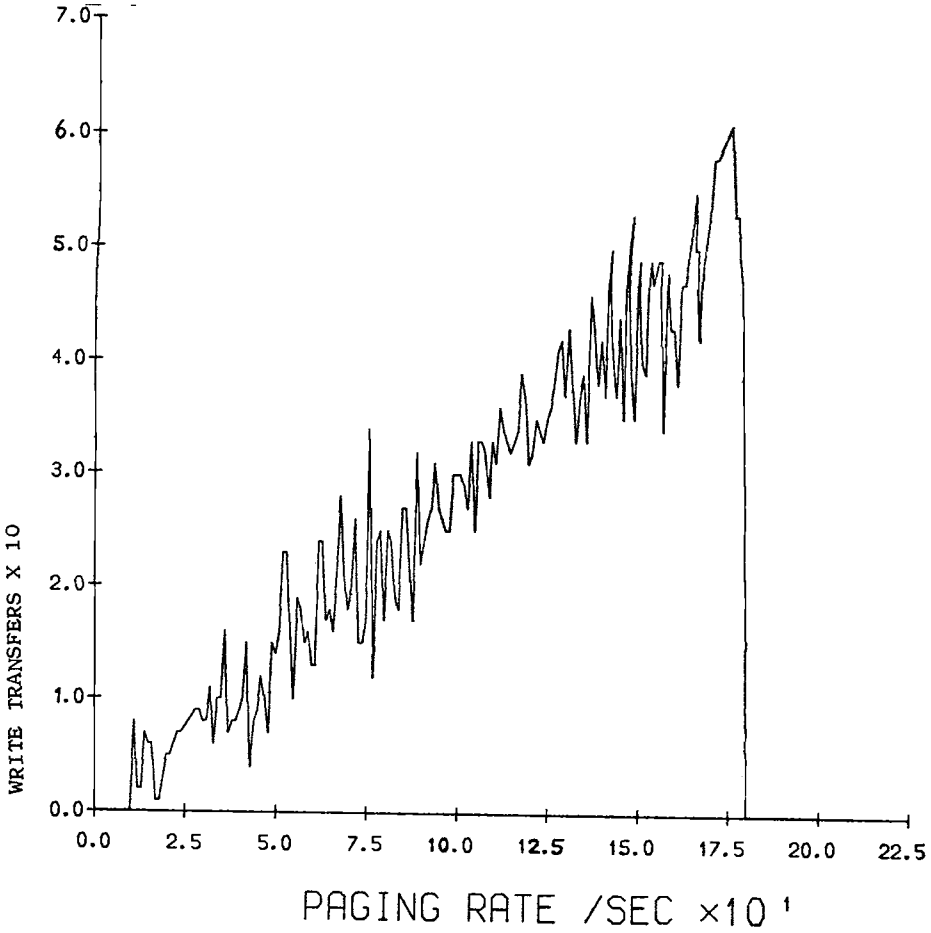


Figure 5.2 (g)

EXPERIMENT A

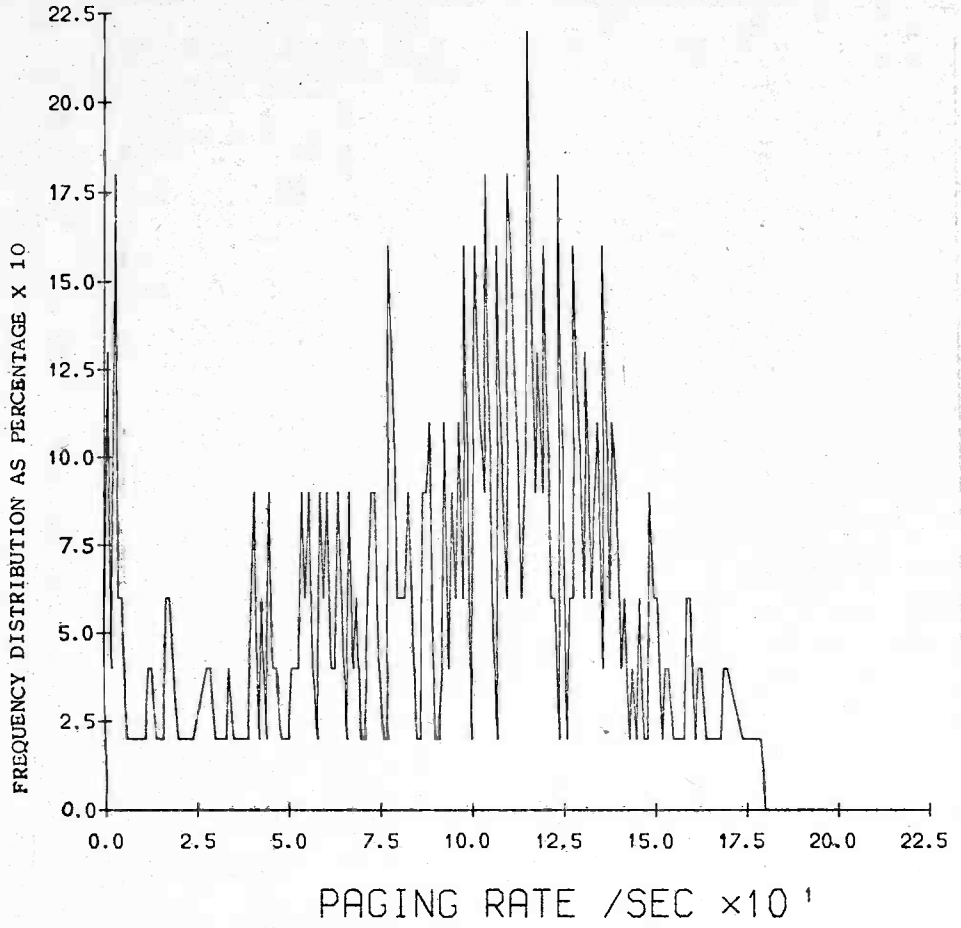


Figure 5.3 (a)

EXPERIMENT B

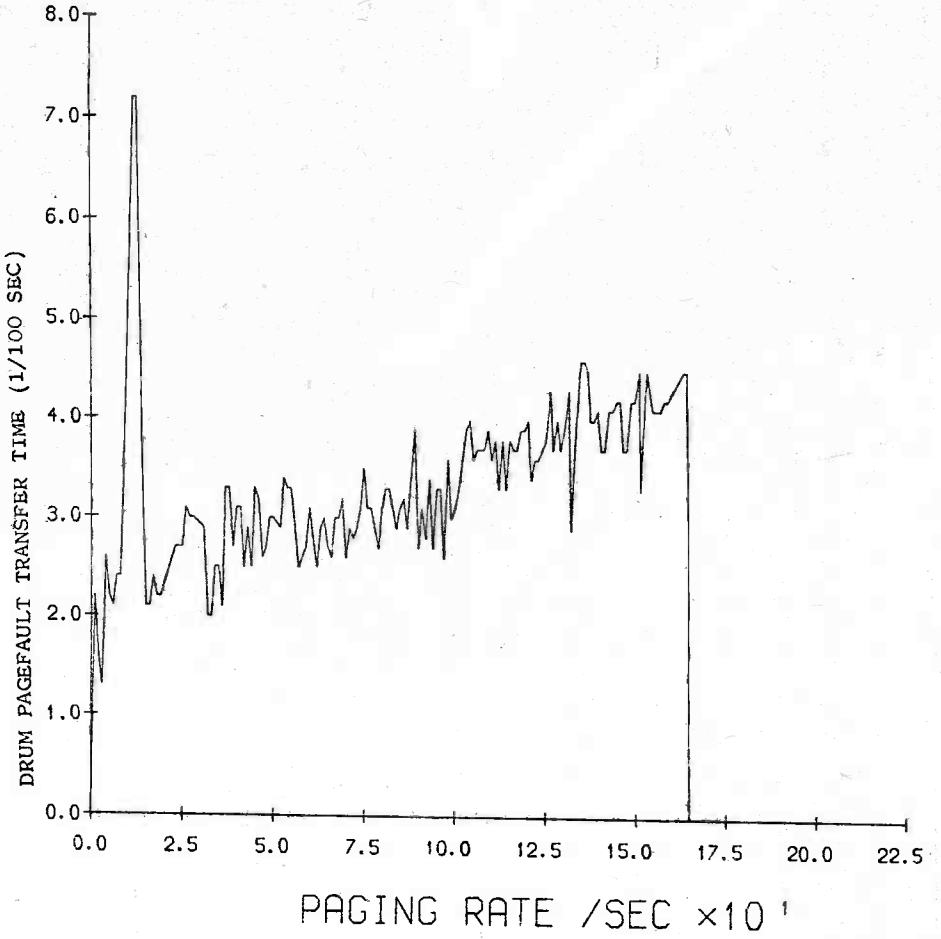


Figure 5.3 (b)

EXPERIMENT B

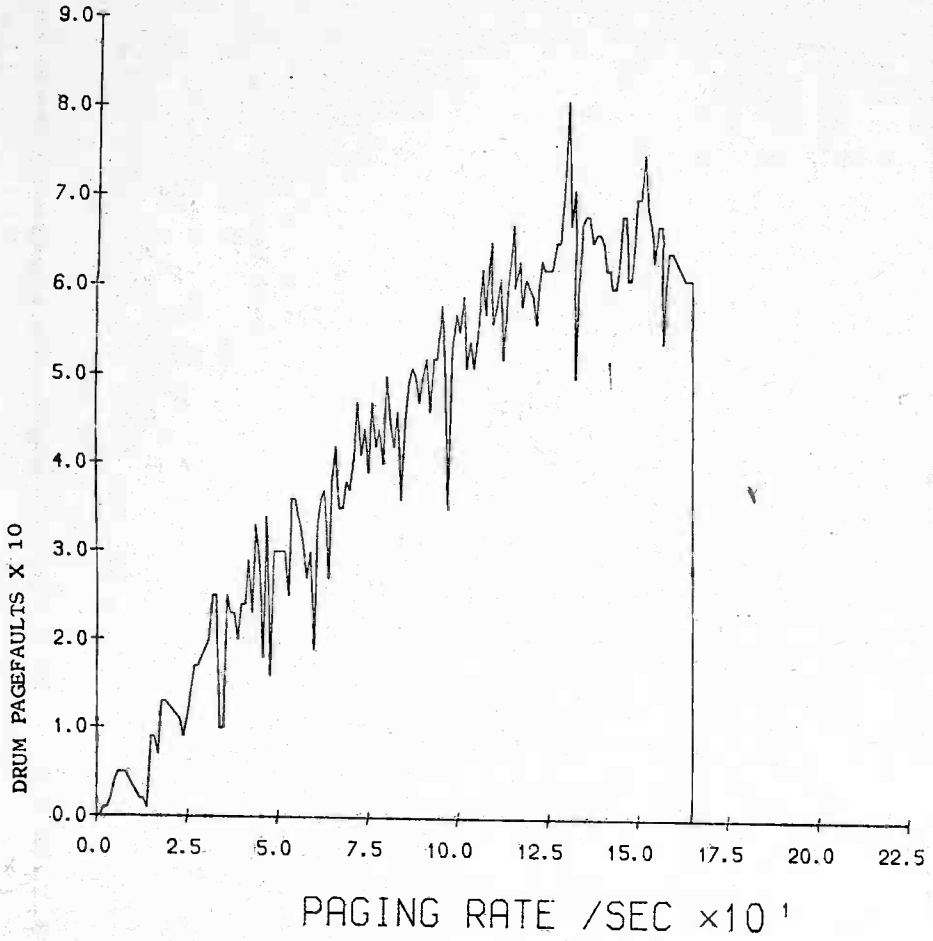


Figure 5.3 (c)

EXPERIMENT B

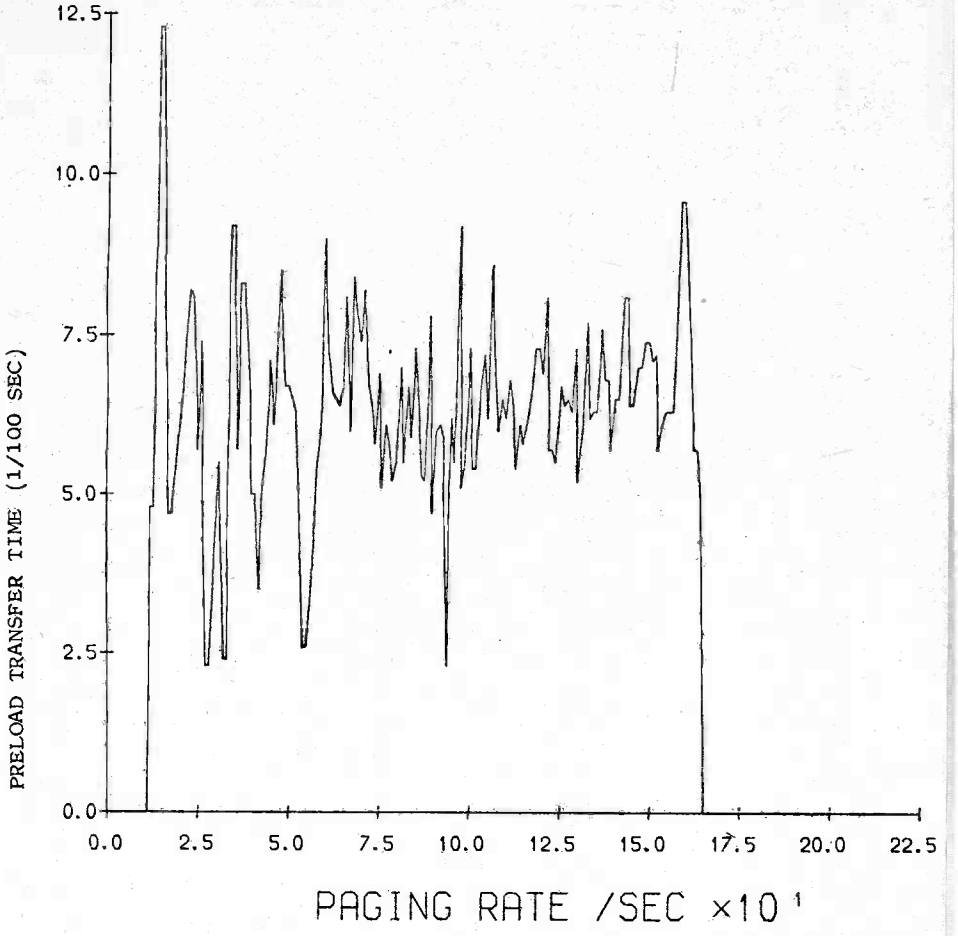


Figure 5.3 (d)

EXPERIMENT B

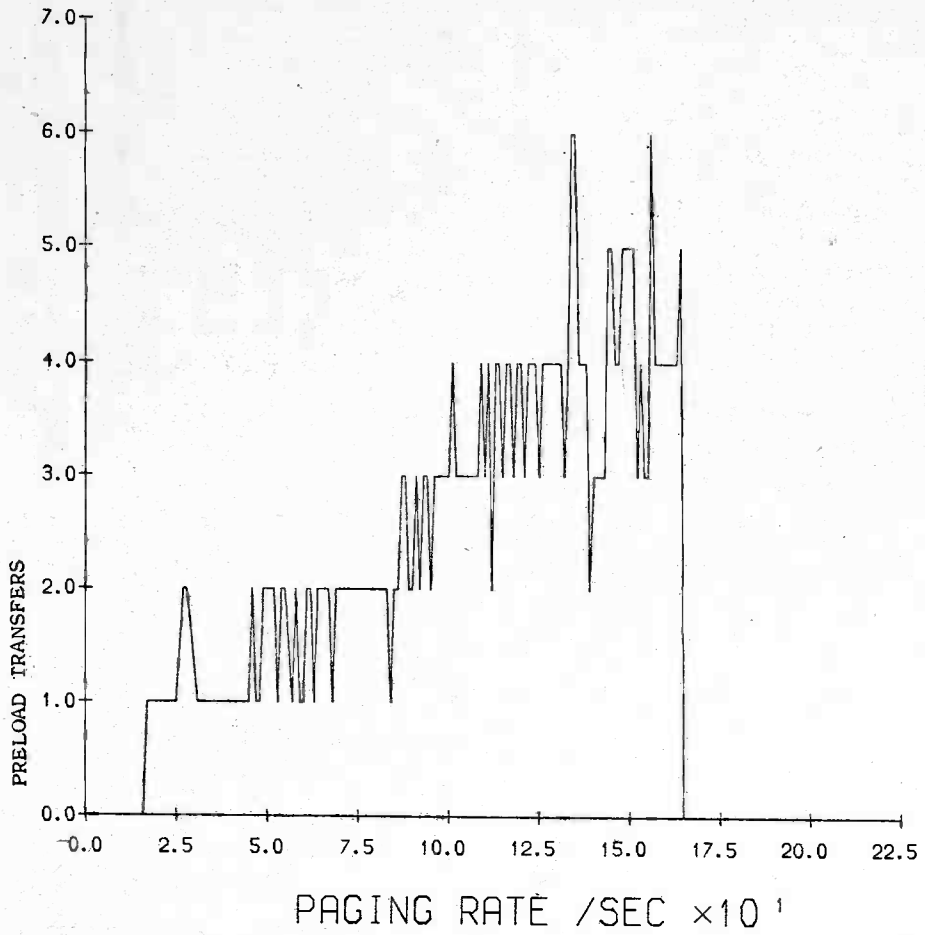


Figure 5.3 (e)

EXPERIMENT B

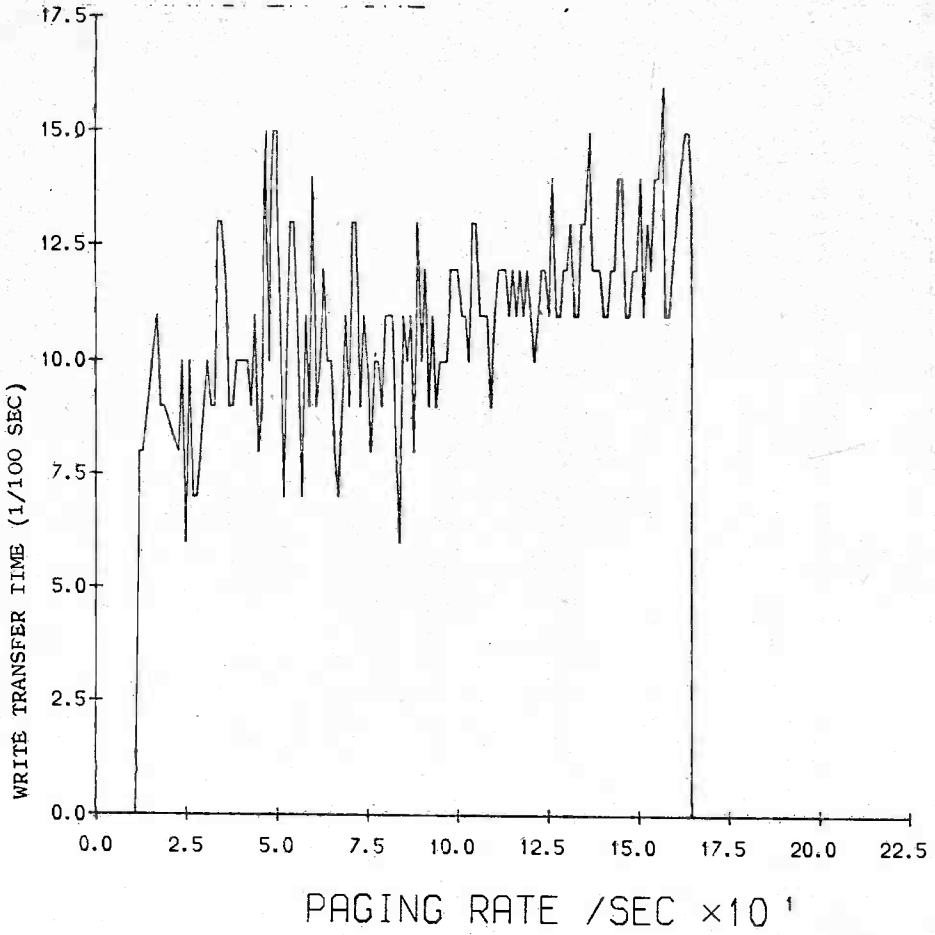


Figure 5.3 (f)

EXPERIMENT B

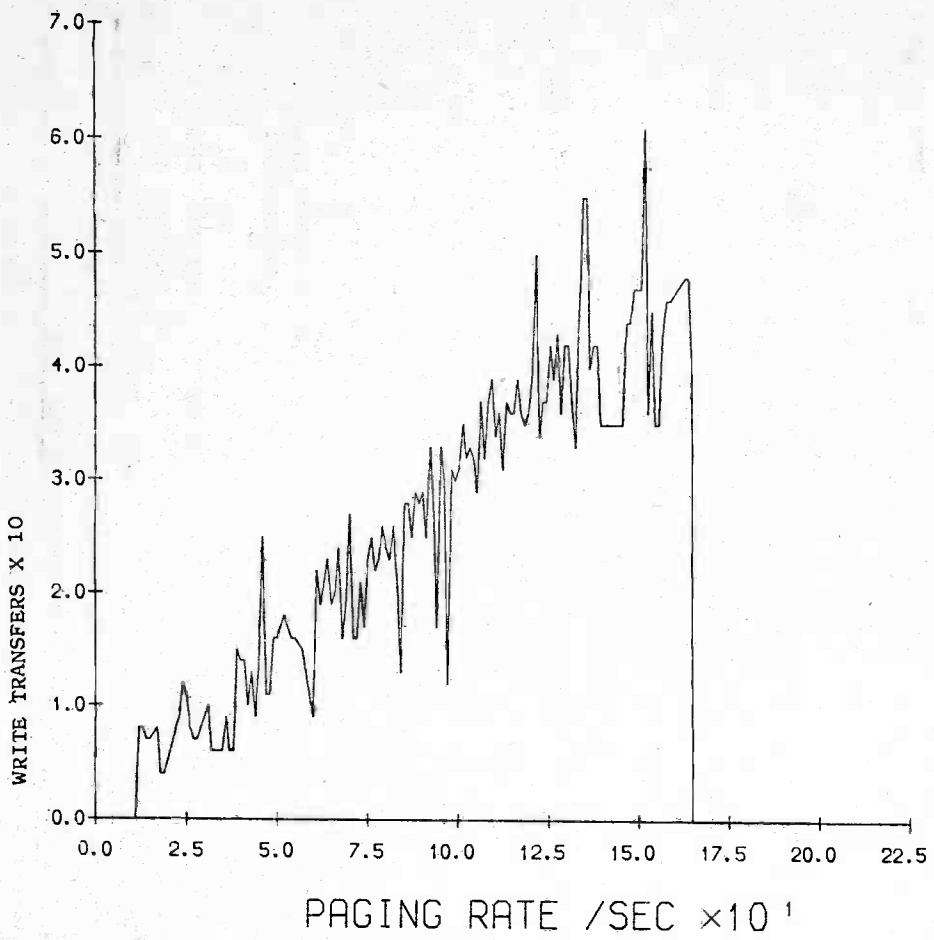


Figure 5.3 (g)

EXPERIMENT B

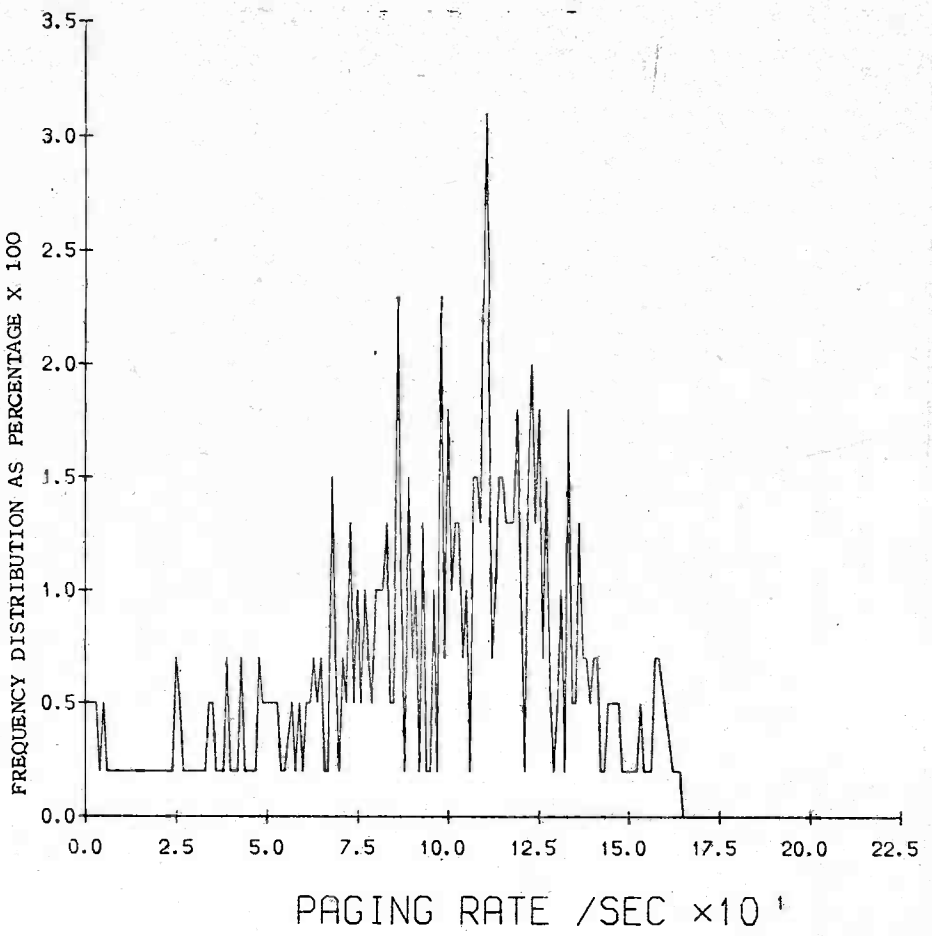


Figure 5.4 (a)

EXPERIMENT C

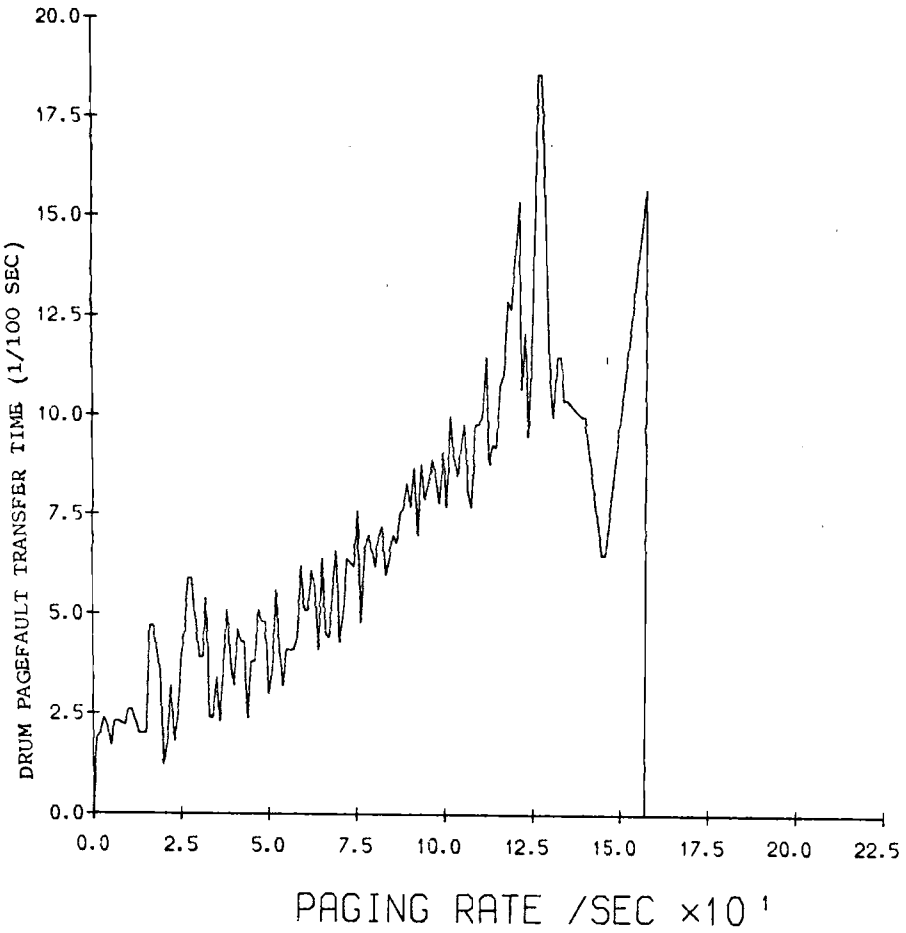


Figure 5.4 (b)

EXPERIMENT C

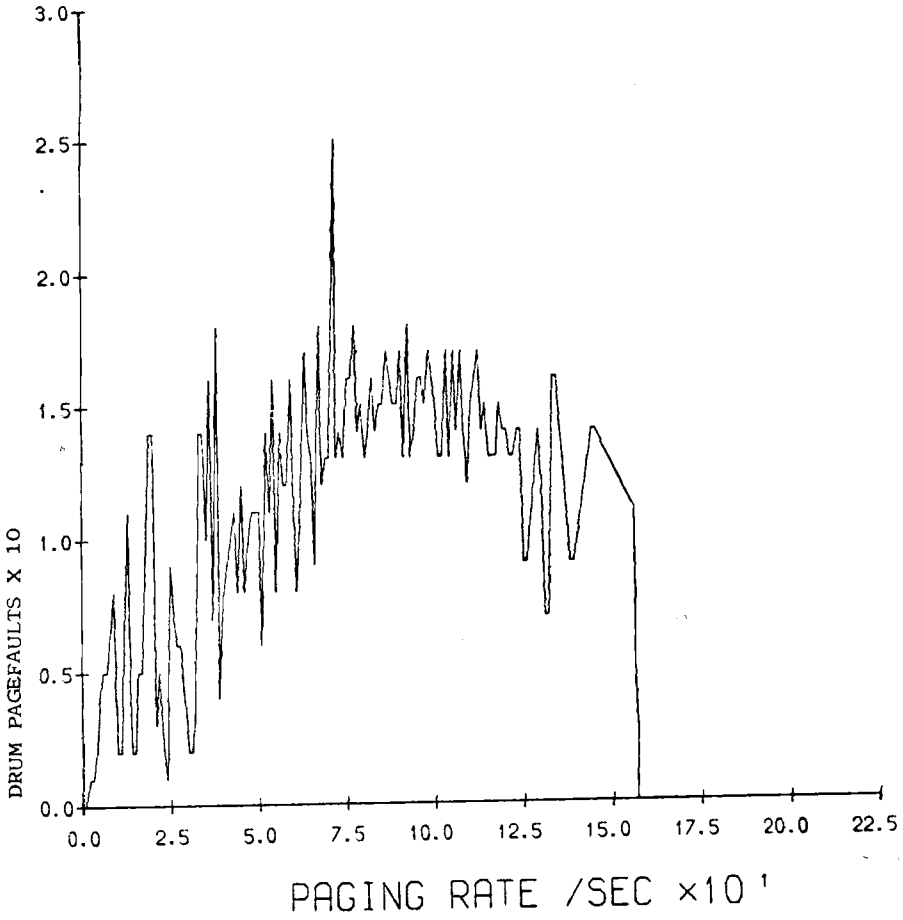


Figure 5.4 (c)

EXPERIMENT C

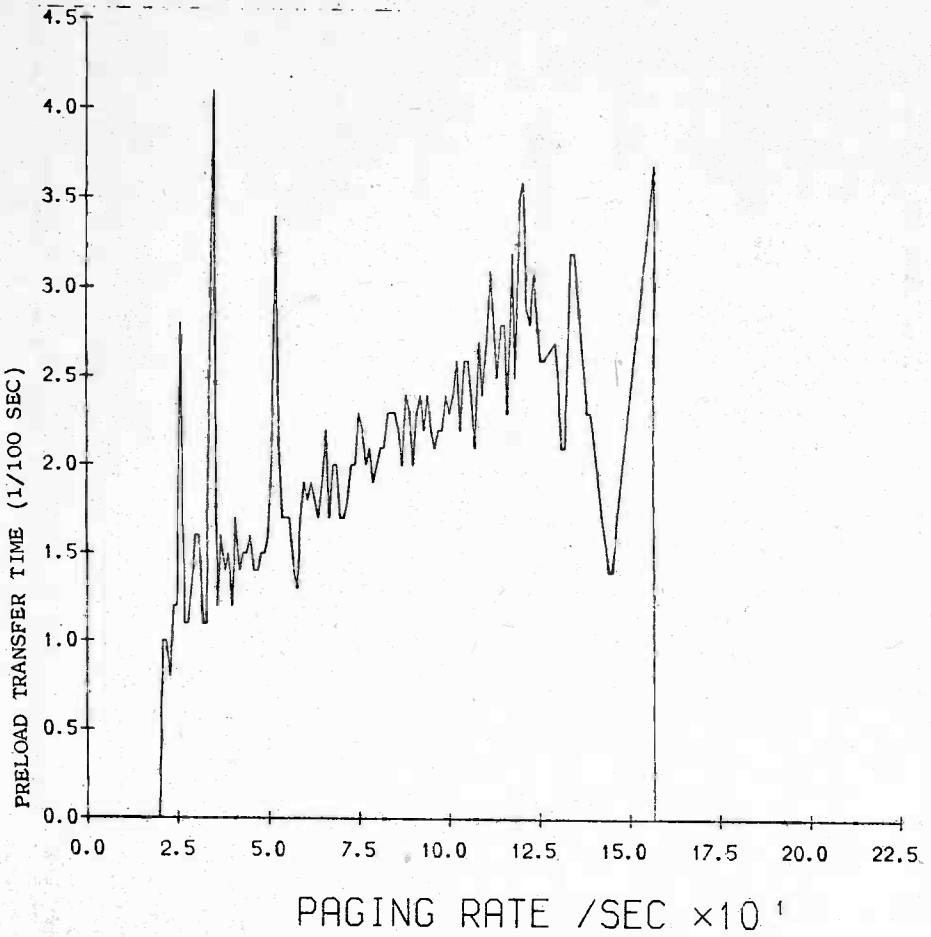


Figure 5.4 (d)

EXPERIMENT C

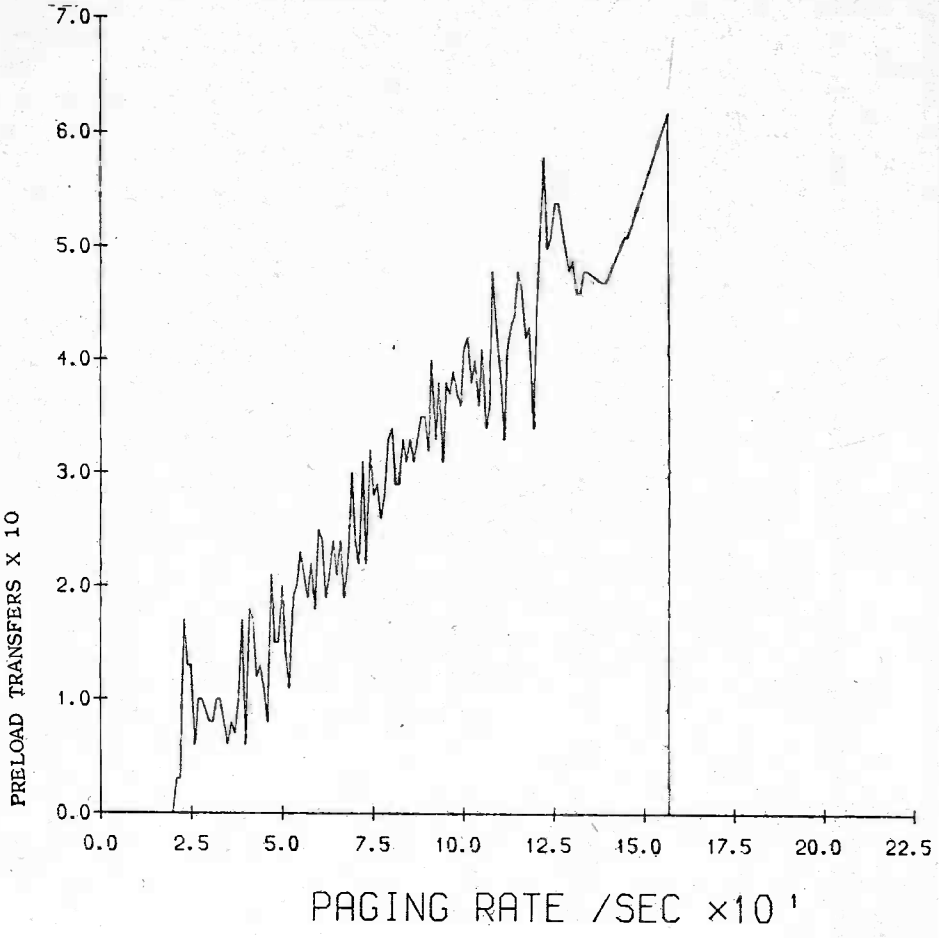


Figure 5.4 (e)

EXPERIMENT C

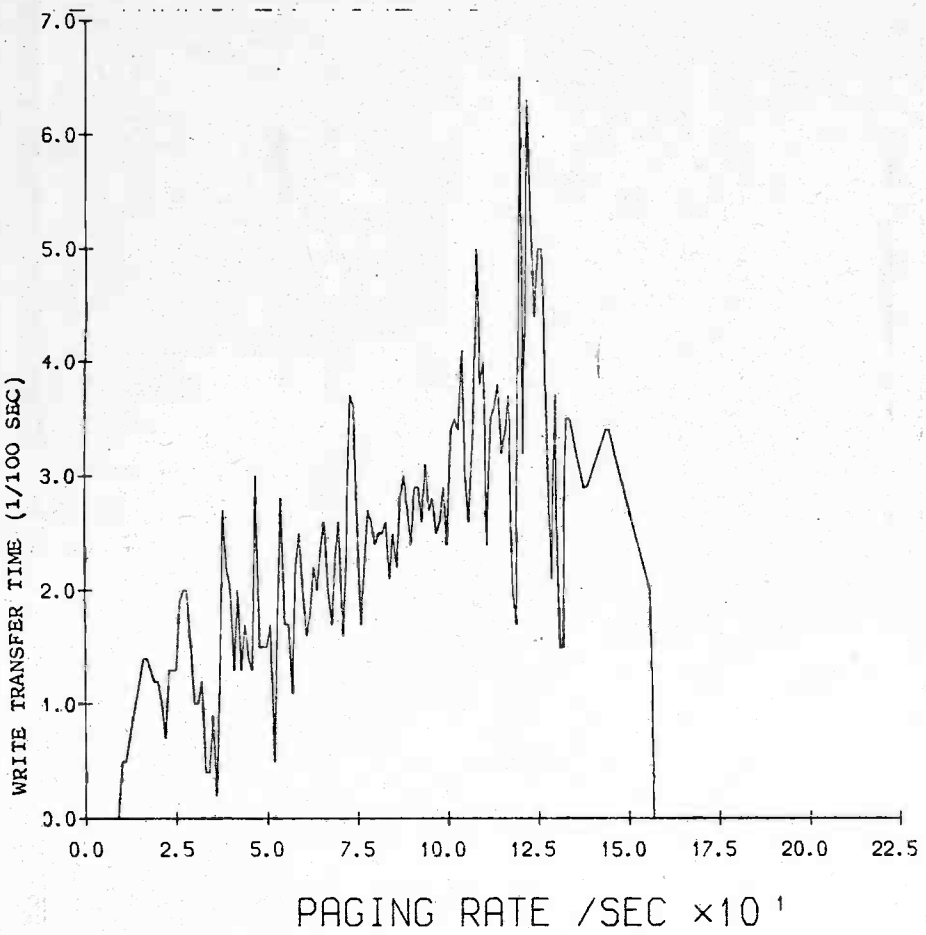


Figure 5.4 (f)

EXPERIMENT C

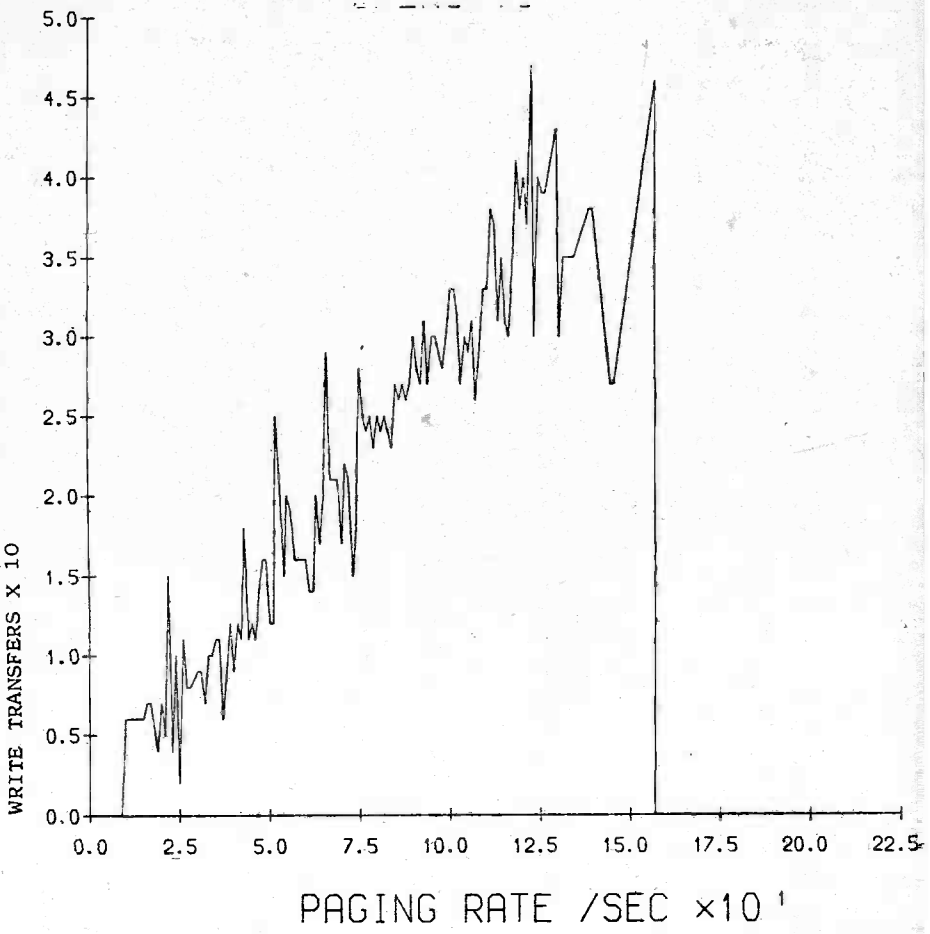


Figure 5.4 (g)

EXPERIMENT C

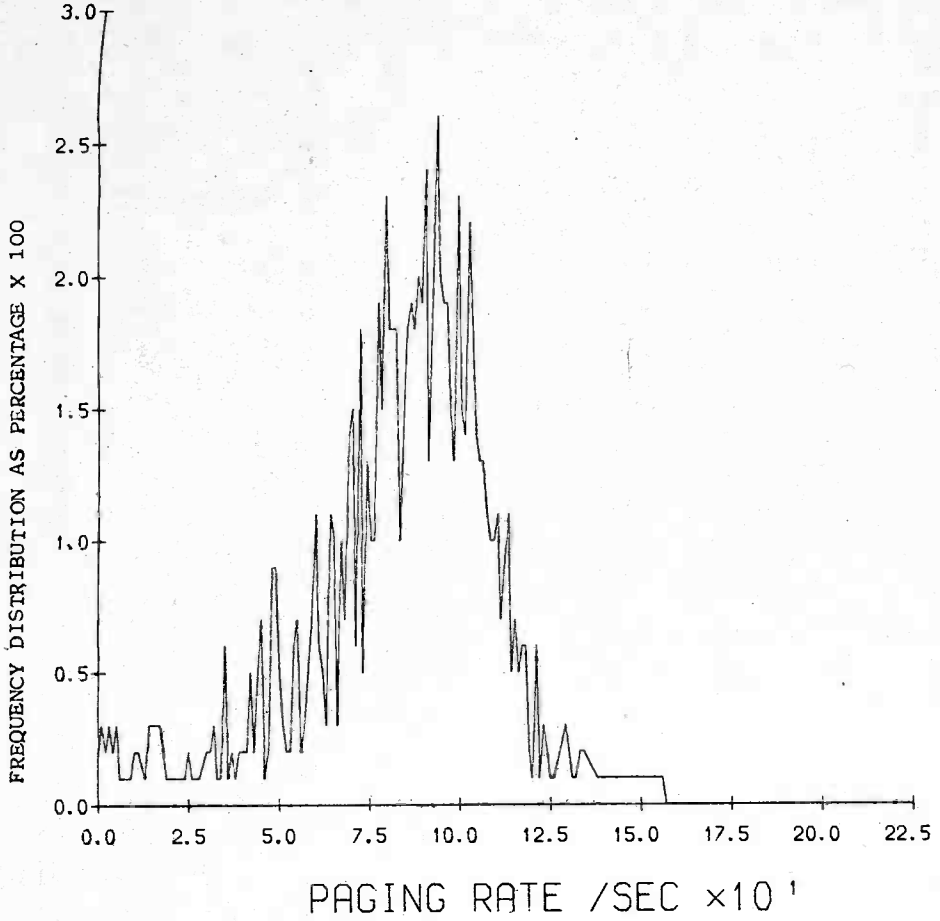


Figure 5.5 (a)

EXPERIMENT D

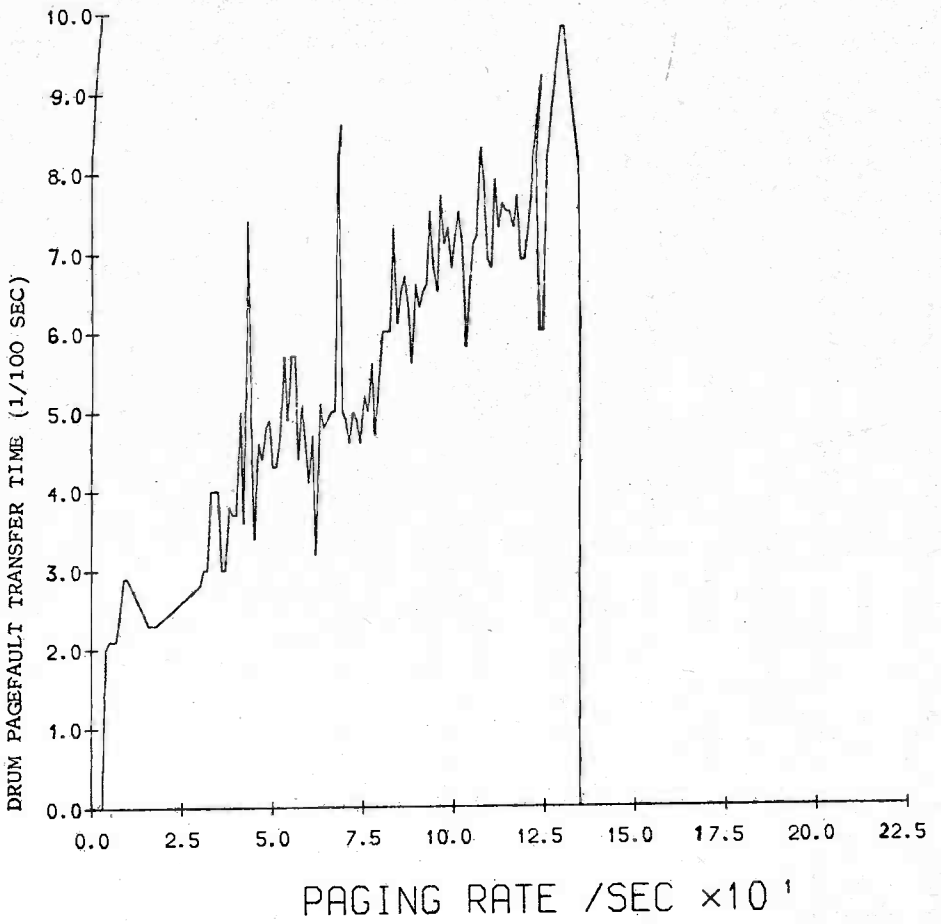


Figure 5.5 (b)

EXPERIMENT D

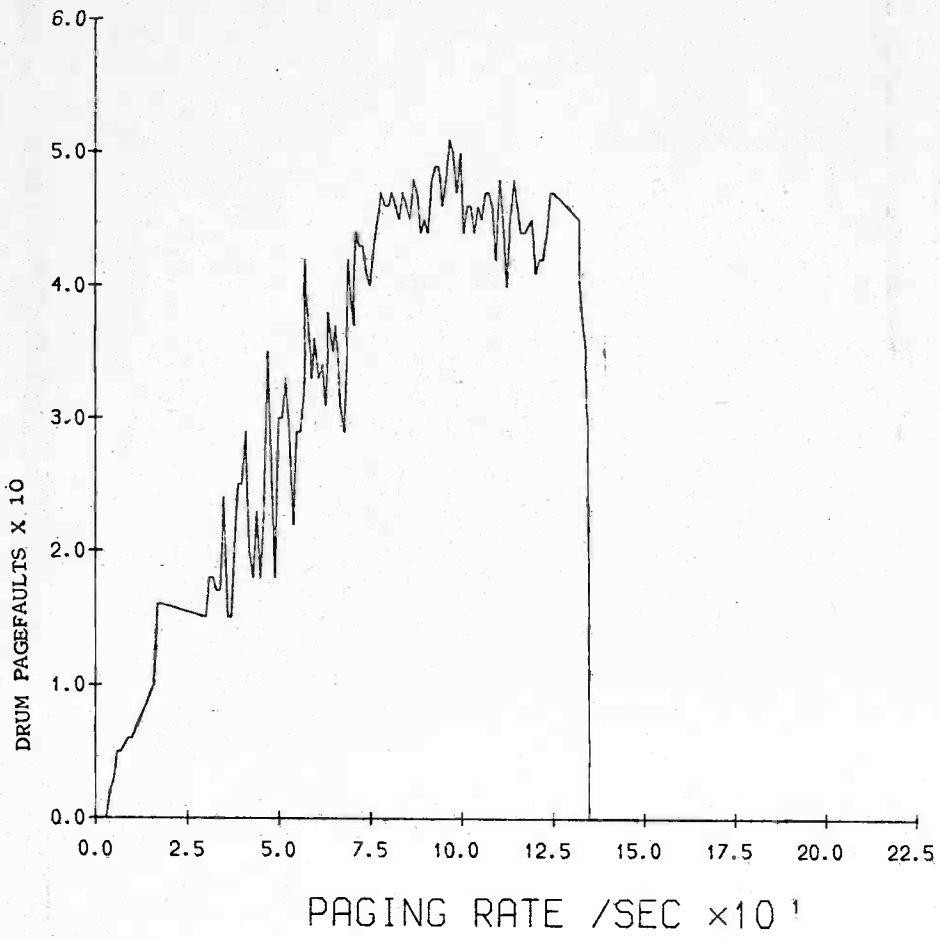


Figure 5.5 (c)

EXPERIMENT D

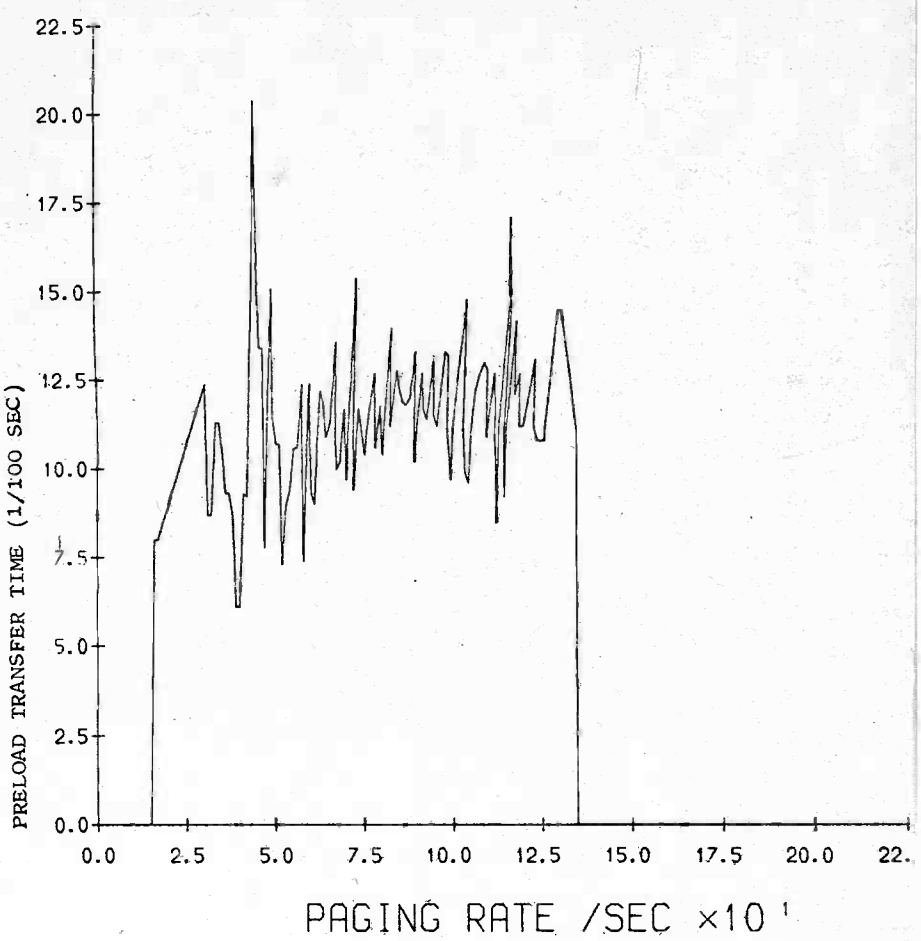


Figure 5.5 (d)

EXPERIMENT D

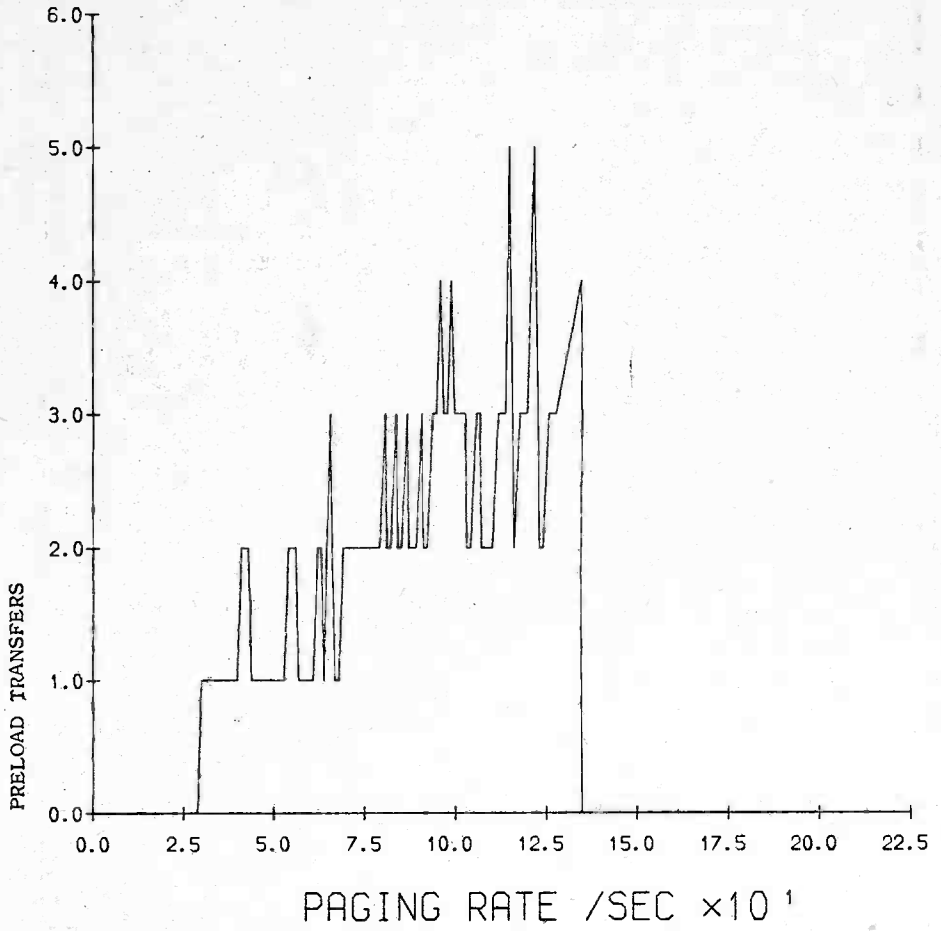


Figure 5.5 (e)

EXPERIMENT D

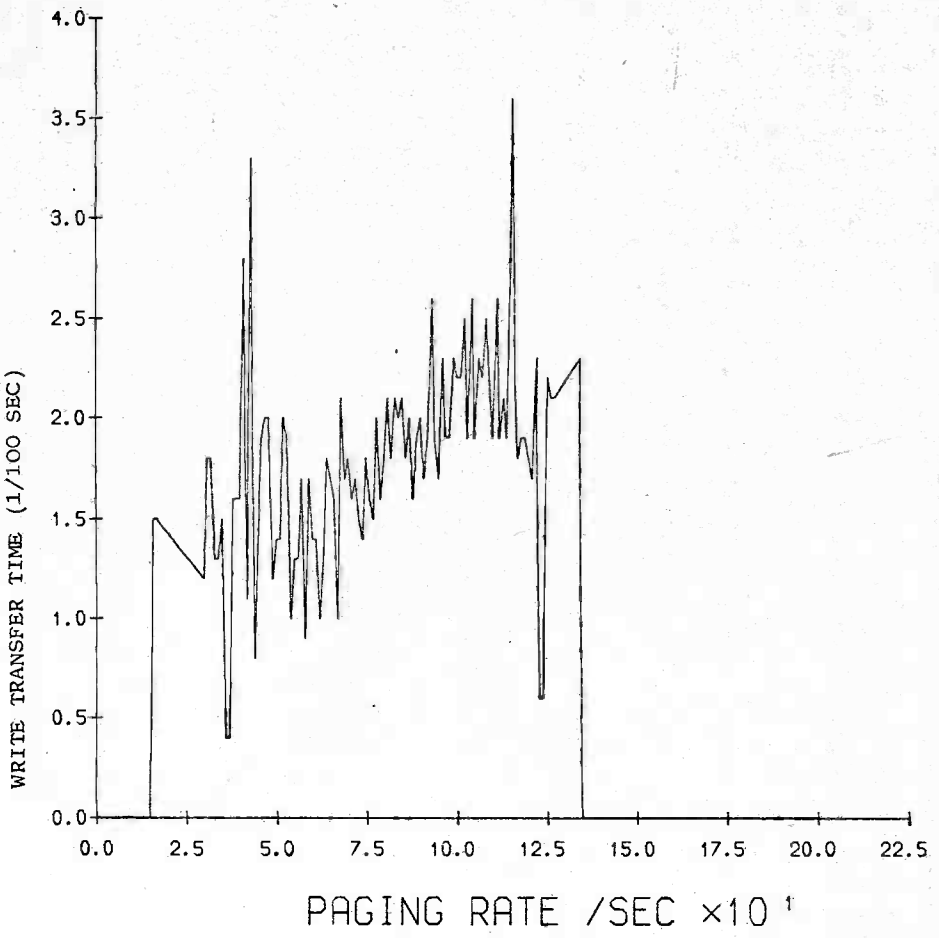


Figure 5.5 (f)

EXPERIMENT D

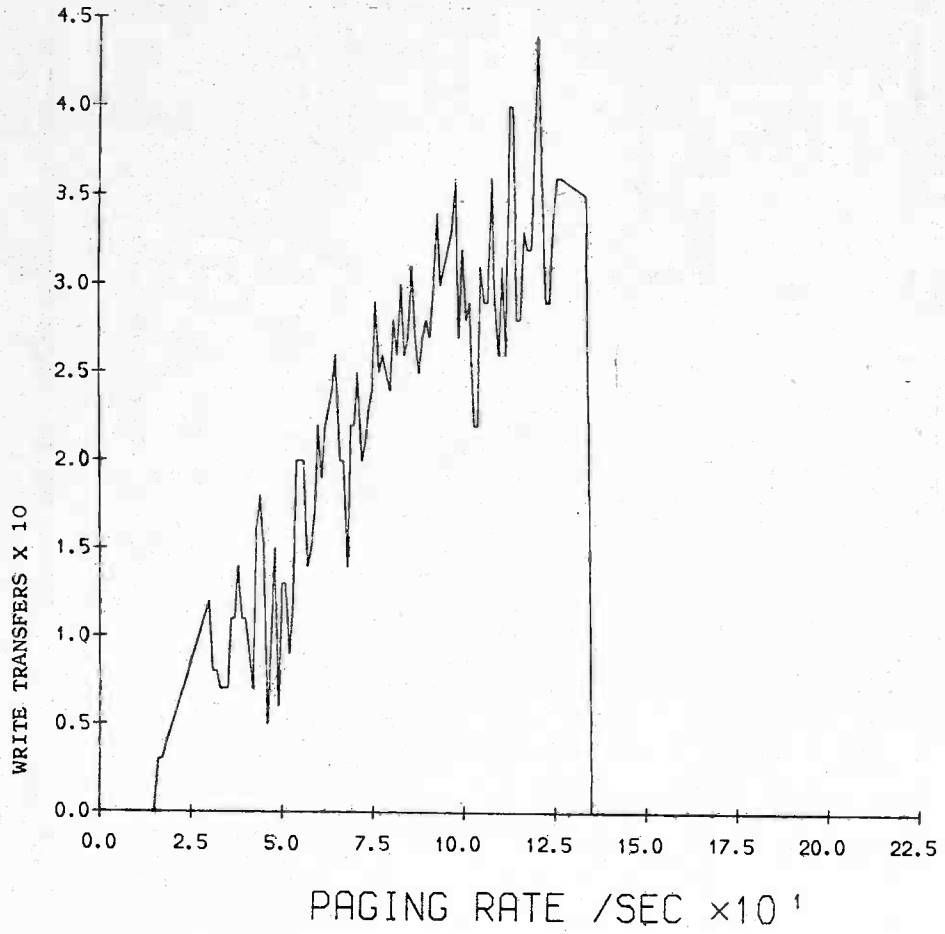


Figure 5.5 (g)

EXPERIMENT D

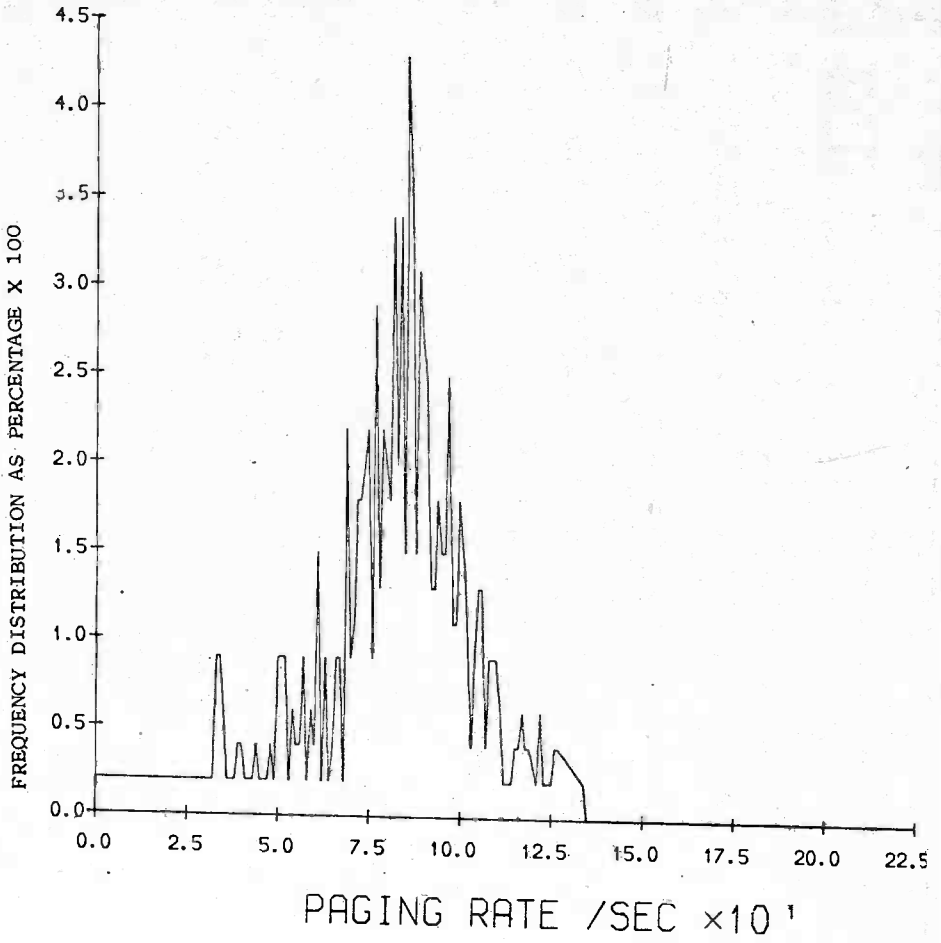


Figure 5.6 (a)

EXPERIMENT E

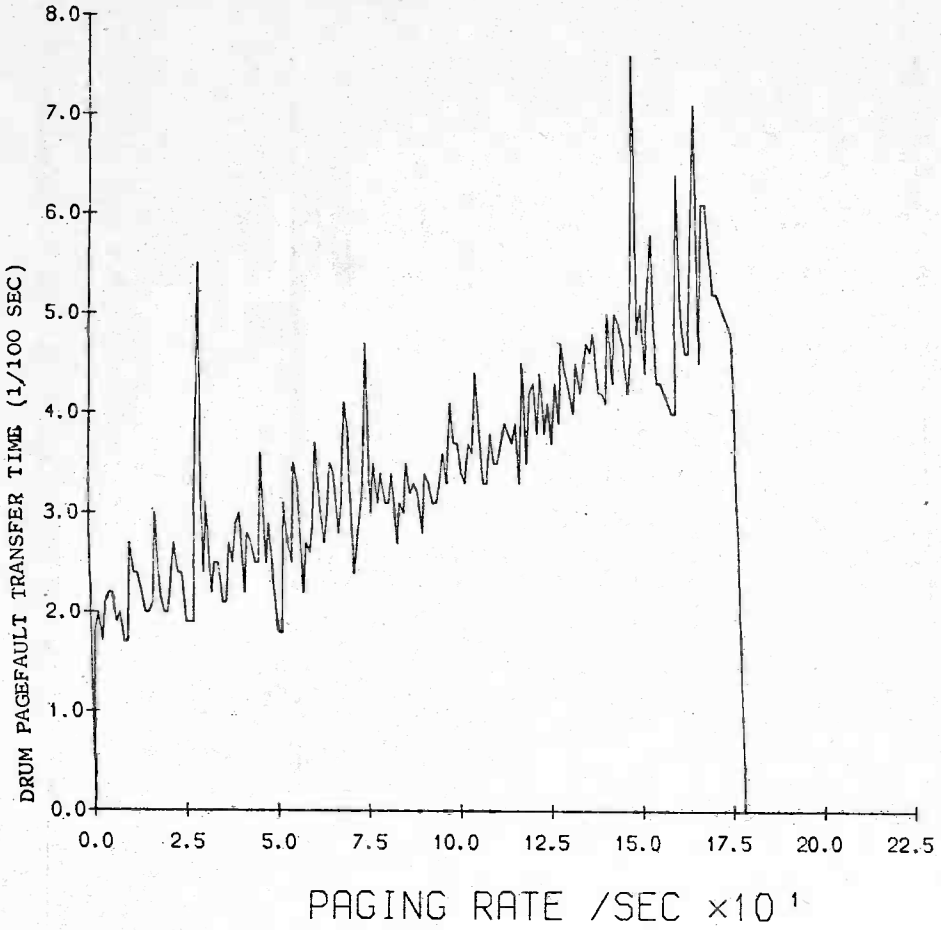


Figure 5.6 (b)

EXPERIMENT E

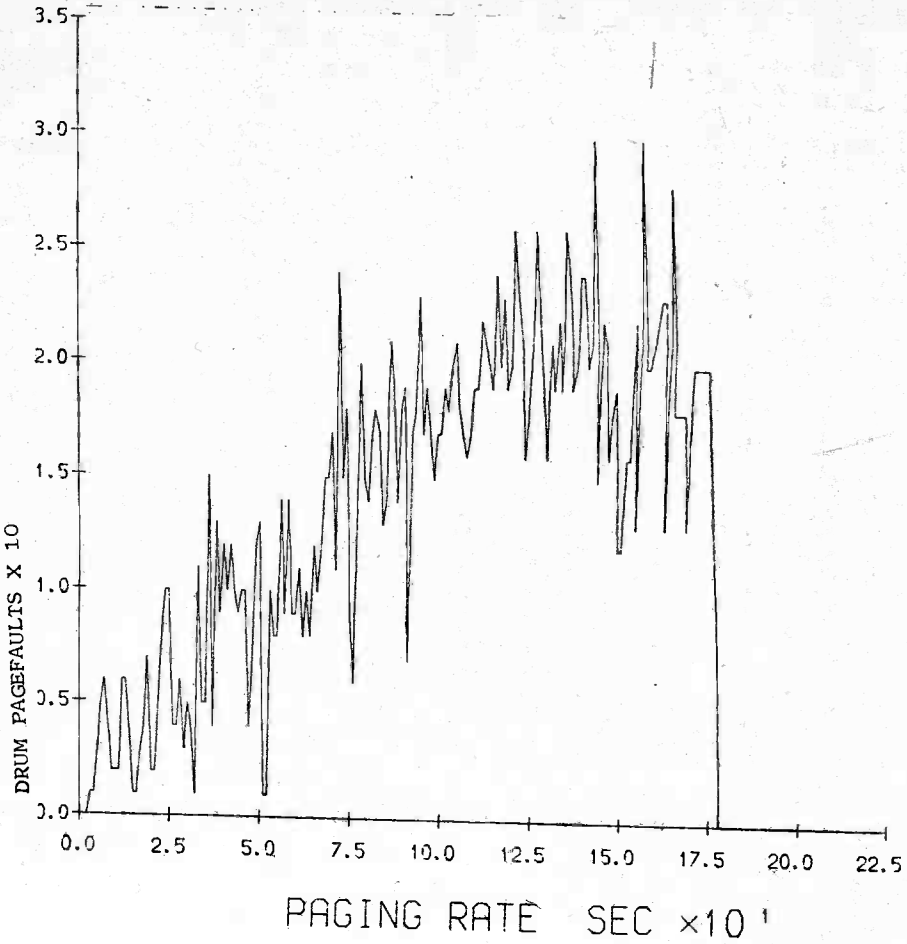


Figure 5.6 (c)

EXPERIMENT E

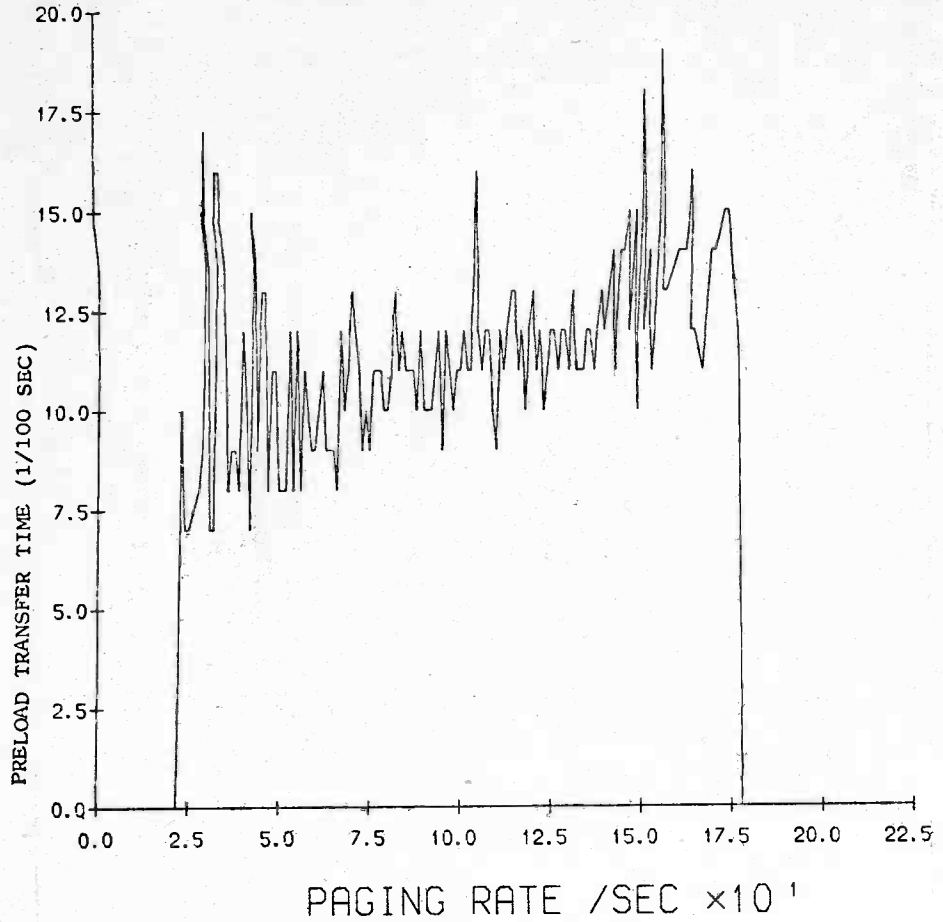


Figure 5.6 (d)

EXPERIMENT E

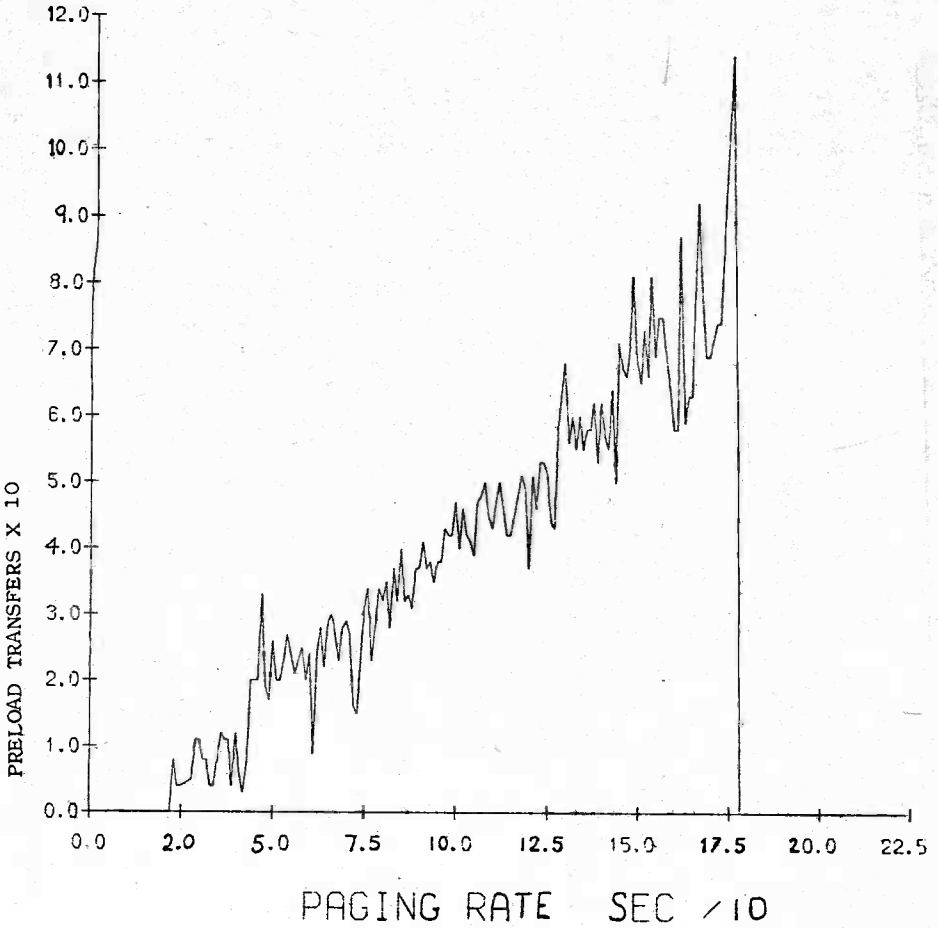


Figure 5.6 (e)

EXPERIMENT E

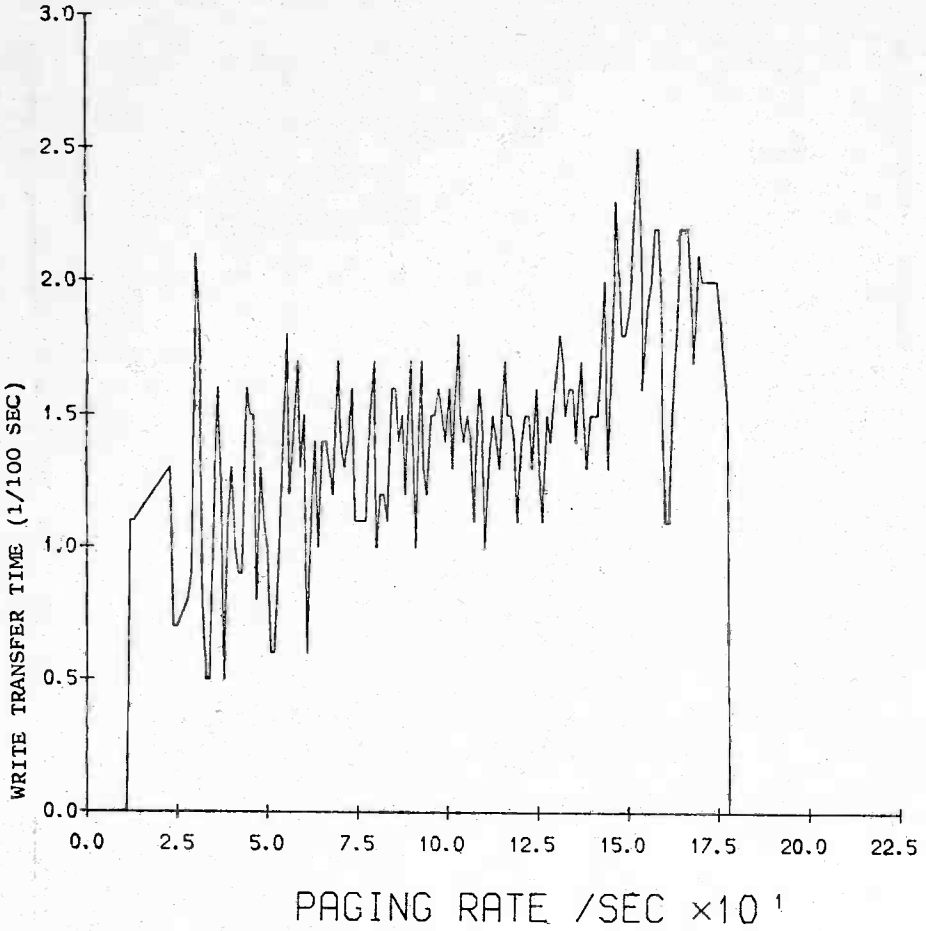


Figure 5.6 (f)

EXPERIMENT E

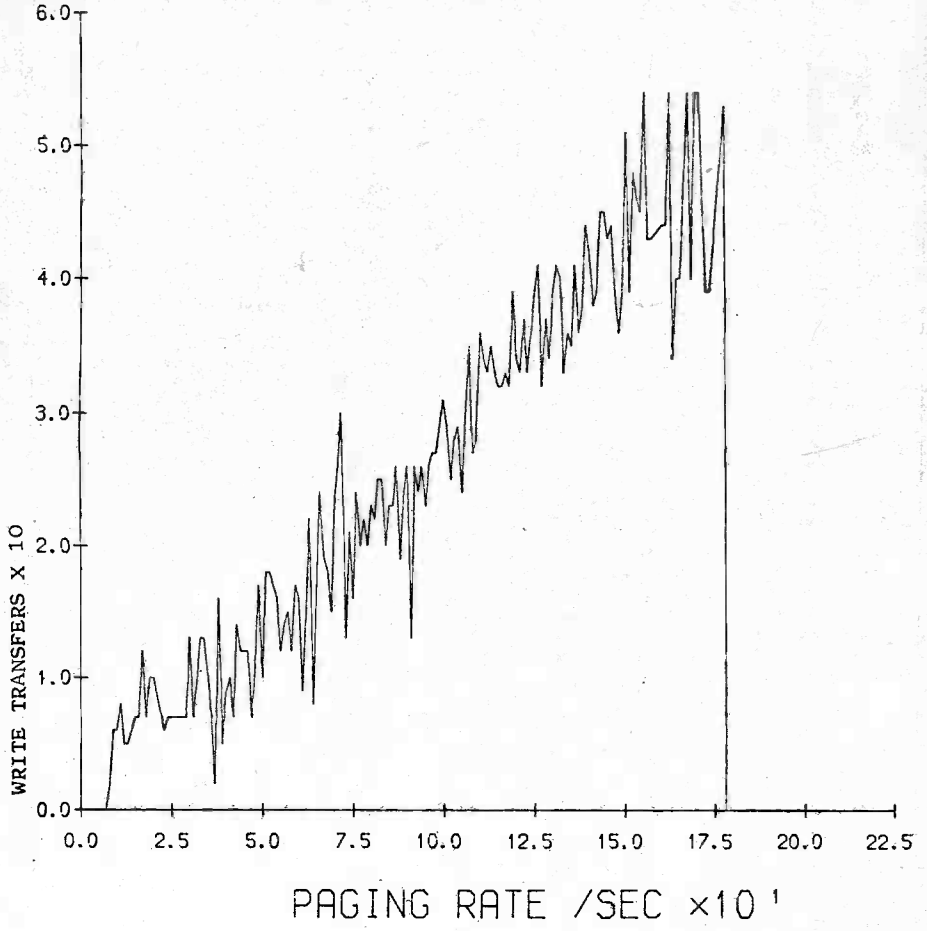


Figure 5.6 (g)

EXPERIMENT E

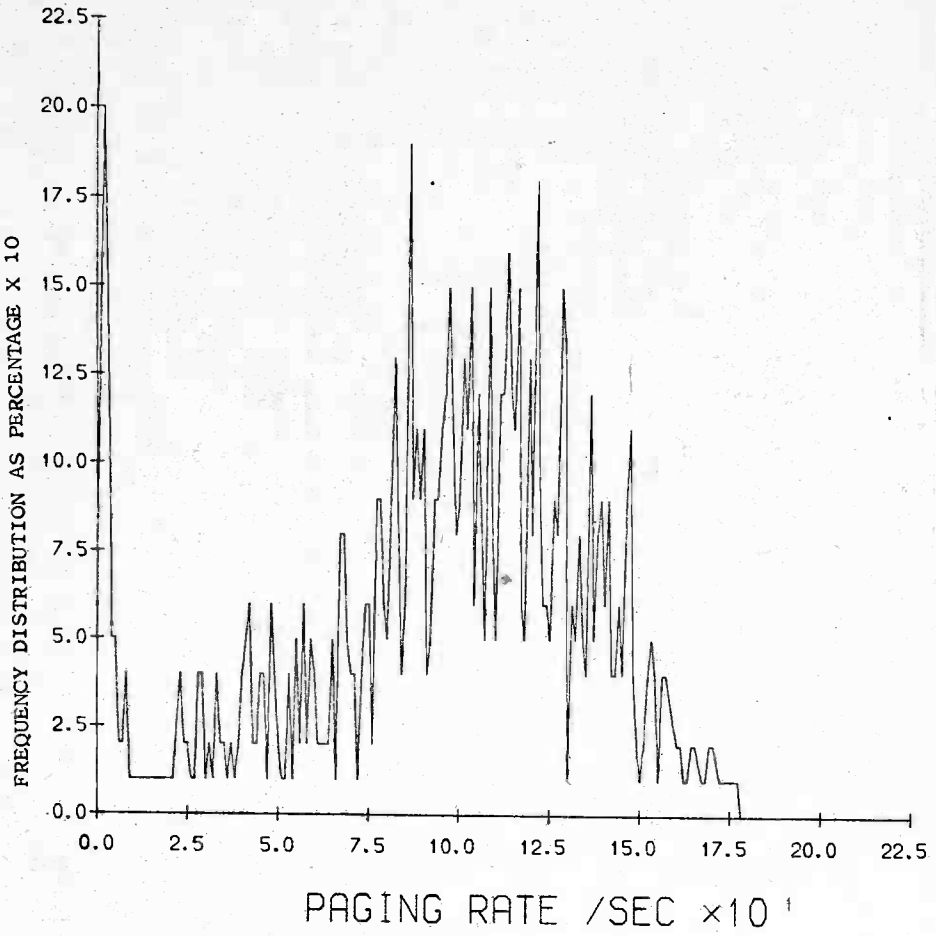


Figure 5.7 (a)

EXPERIMENT F

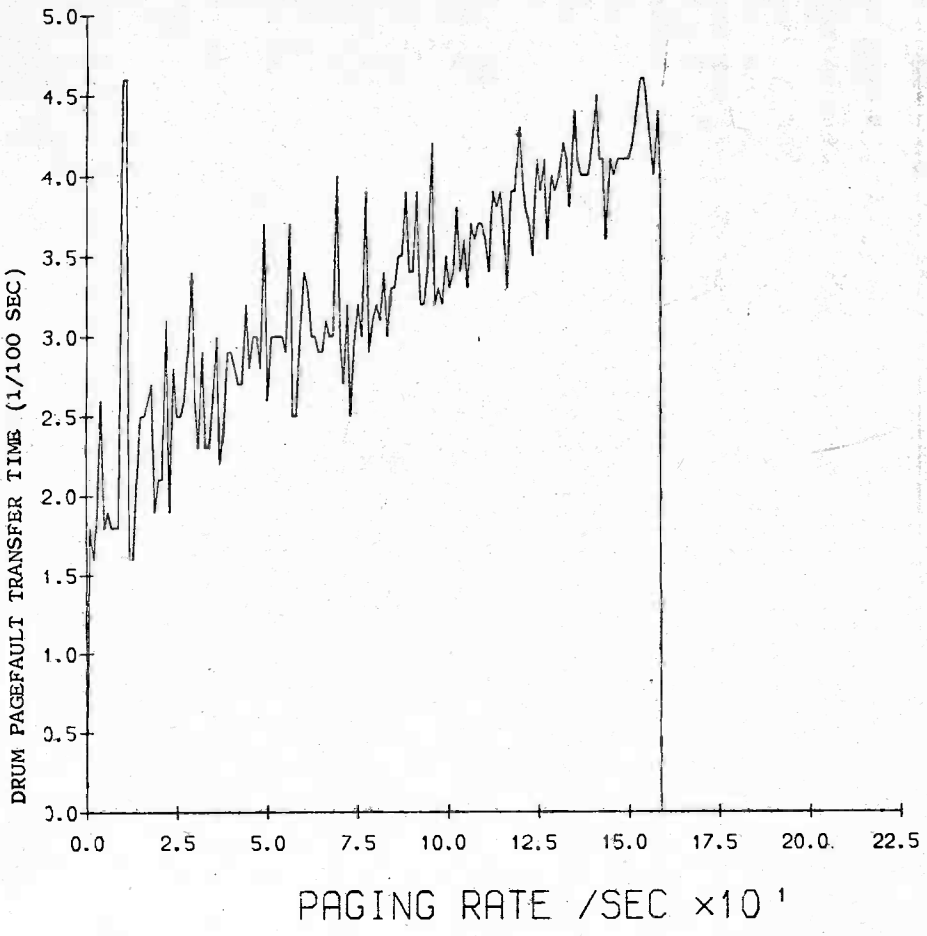


Figure 5.7 (b)

EXPERIMENT F

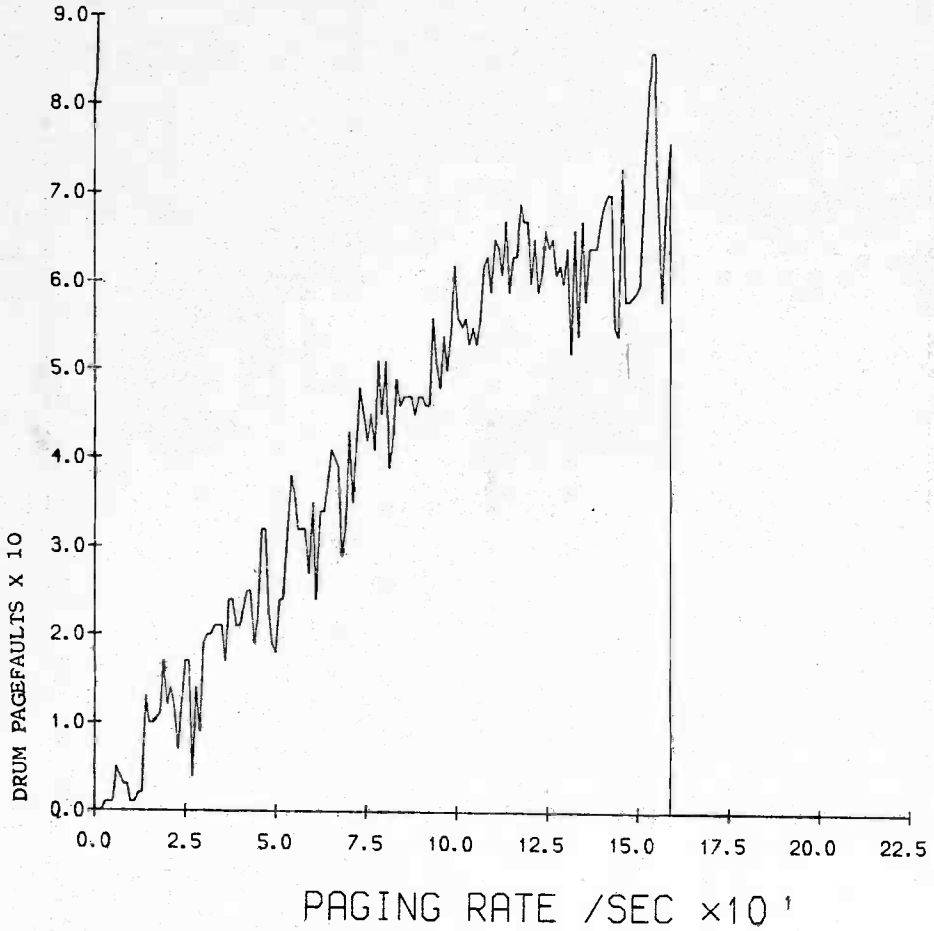


Figure 5.7 (c)

EXPERIMENT F

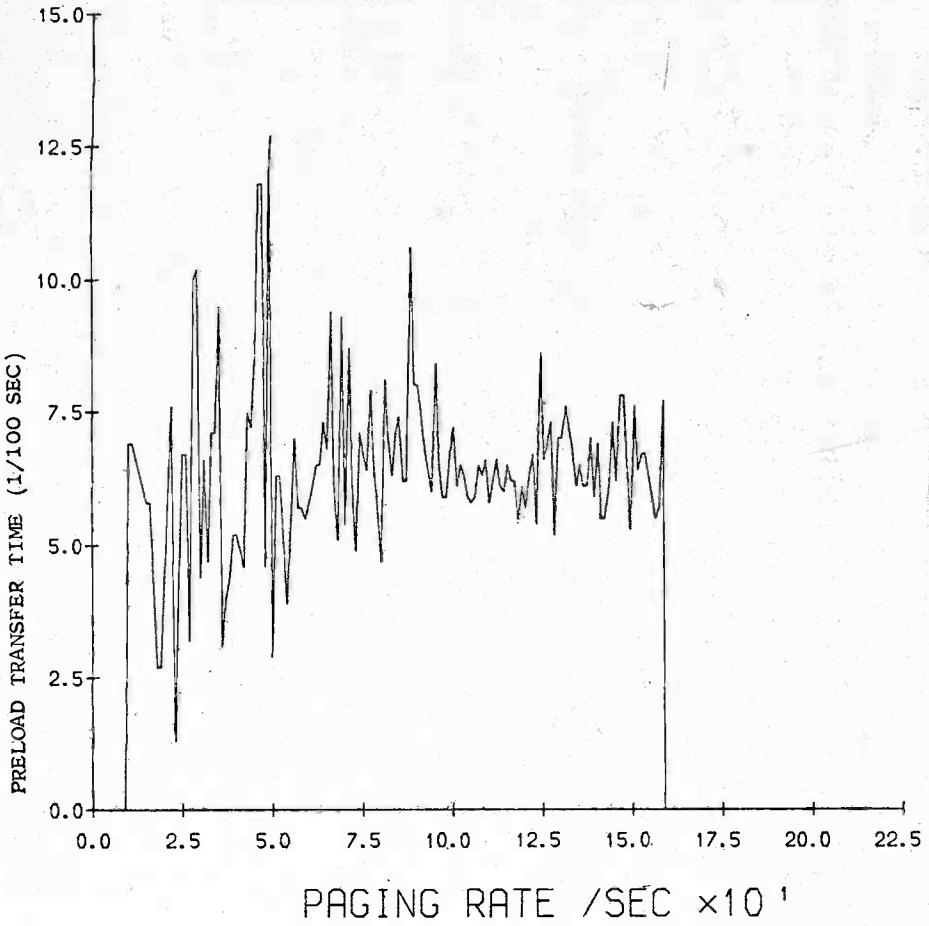


Figure 5.7 (d)

EXPERIMENT F

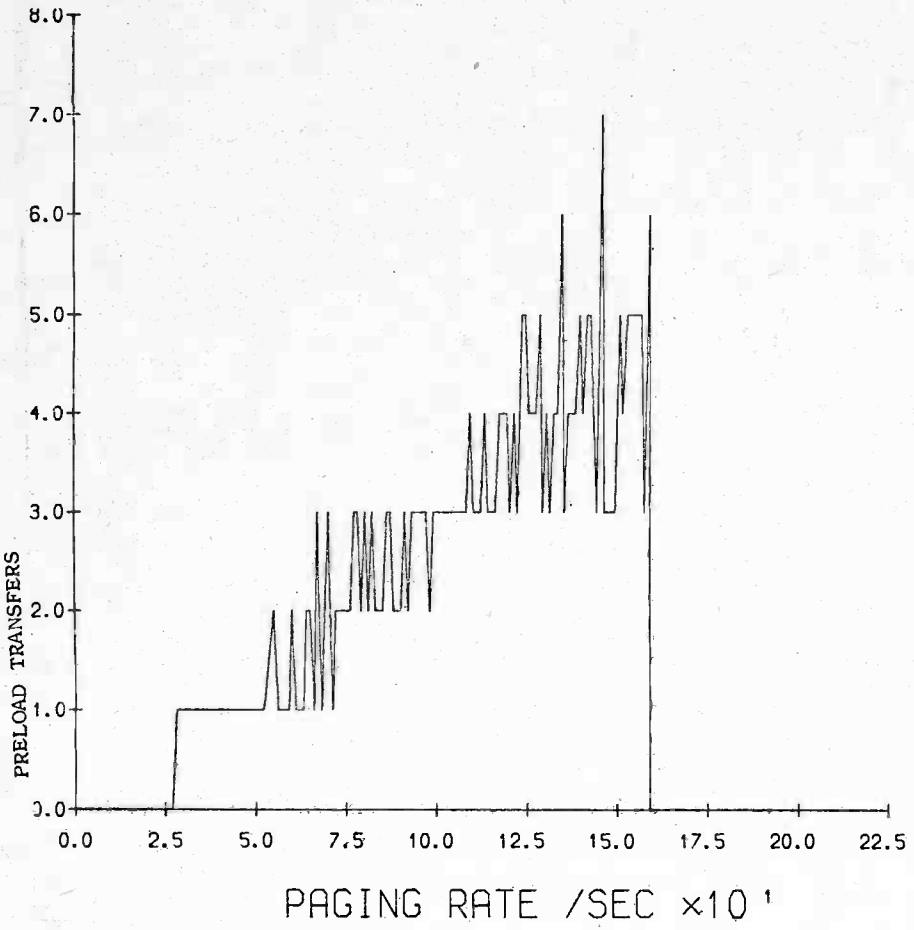


Figure 5.7 (e)

EXPERIMENT F

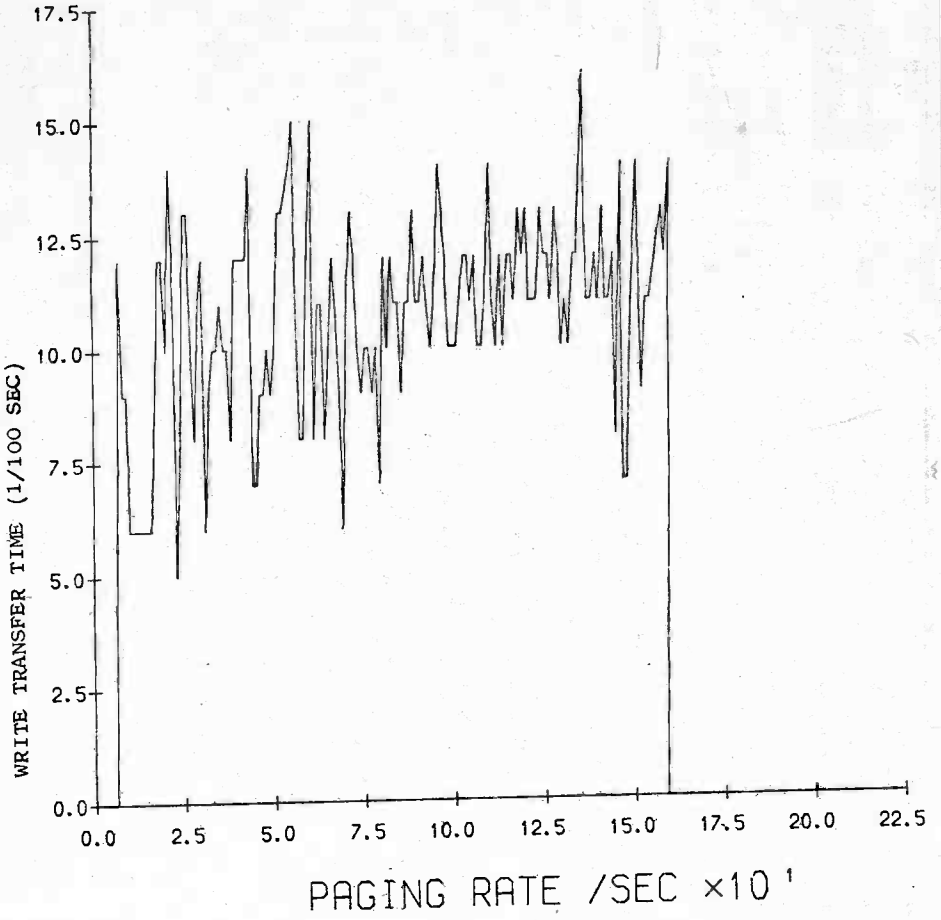


Figure 5.7 (f)

EXPERIMENT F

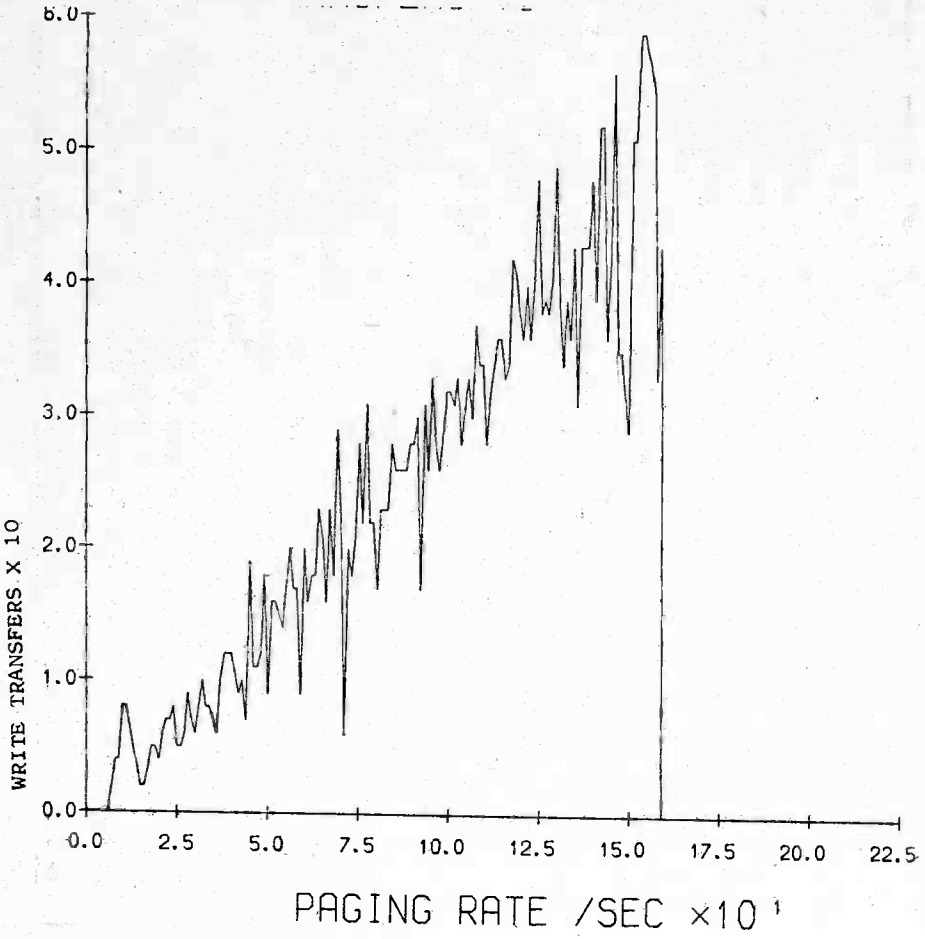


Figure 5.7 (g)

EXPERIMENT F

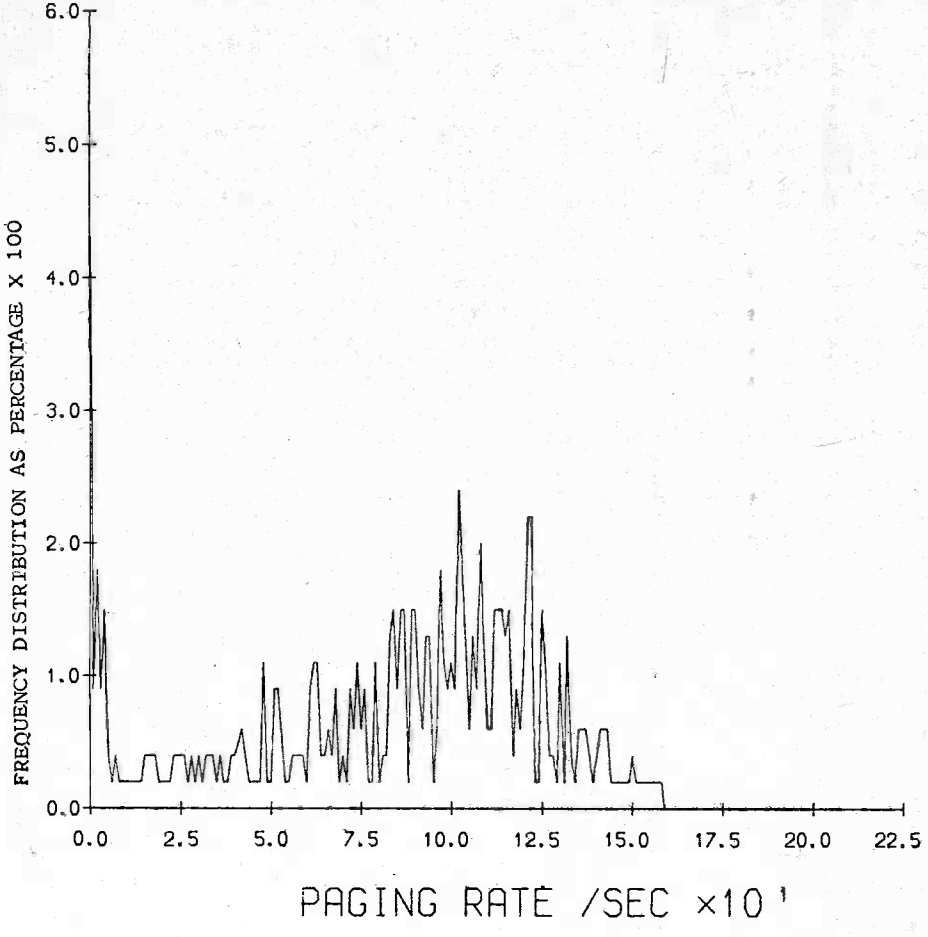


Figure 5.8 (a)

EXPERIMENT G

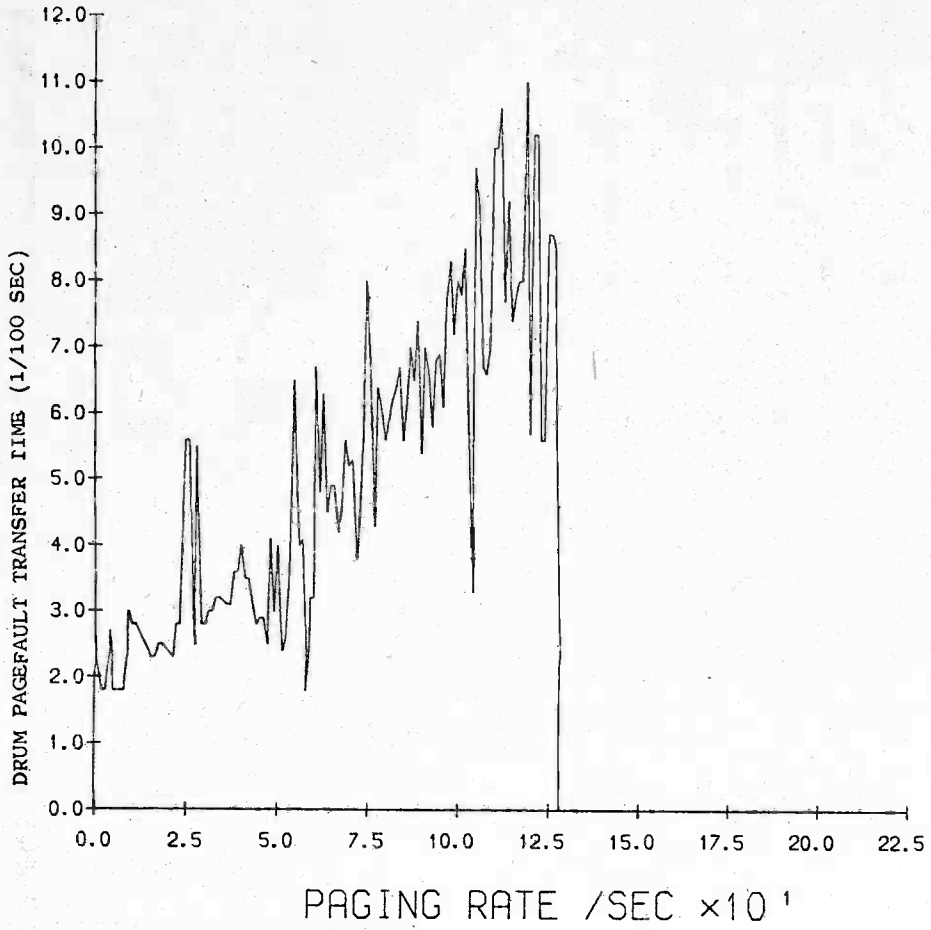


Figure 5.8 (b)

EXPERIMENT G

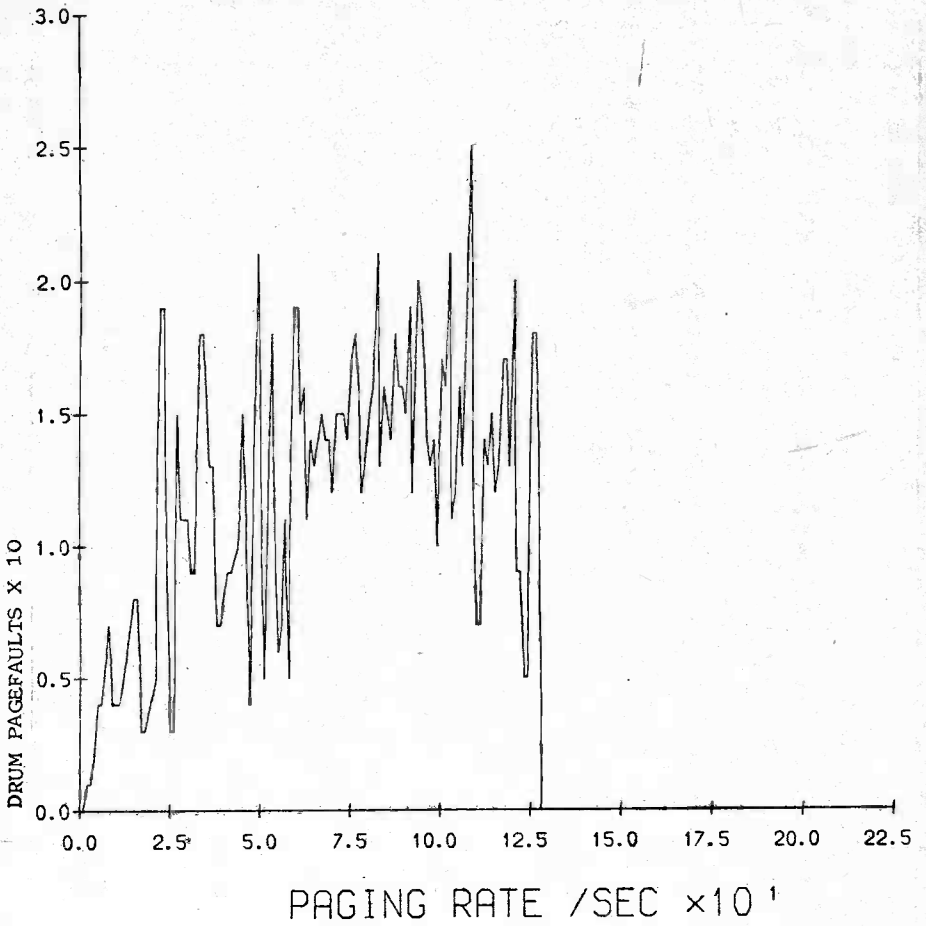


Figure 5.8 (c)

EXPERIMENT G

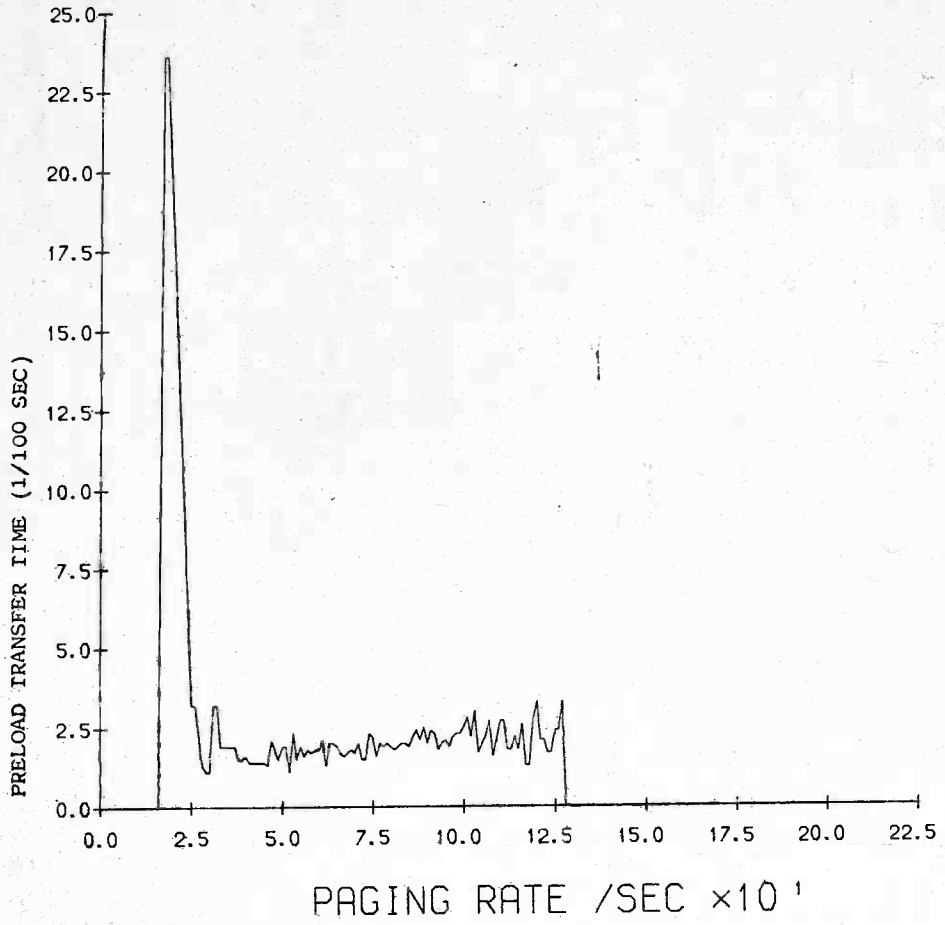


Figure 5.8 (d)

EXPERIMENT G

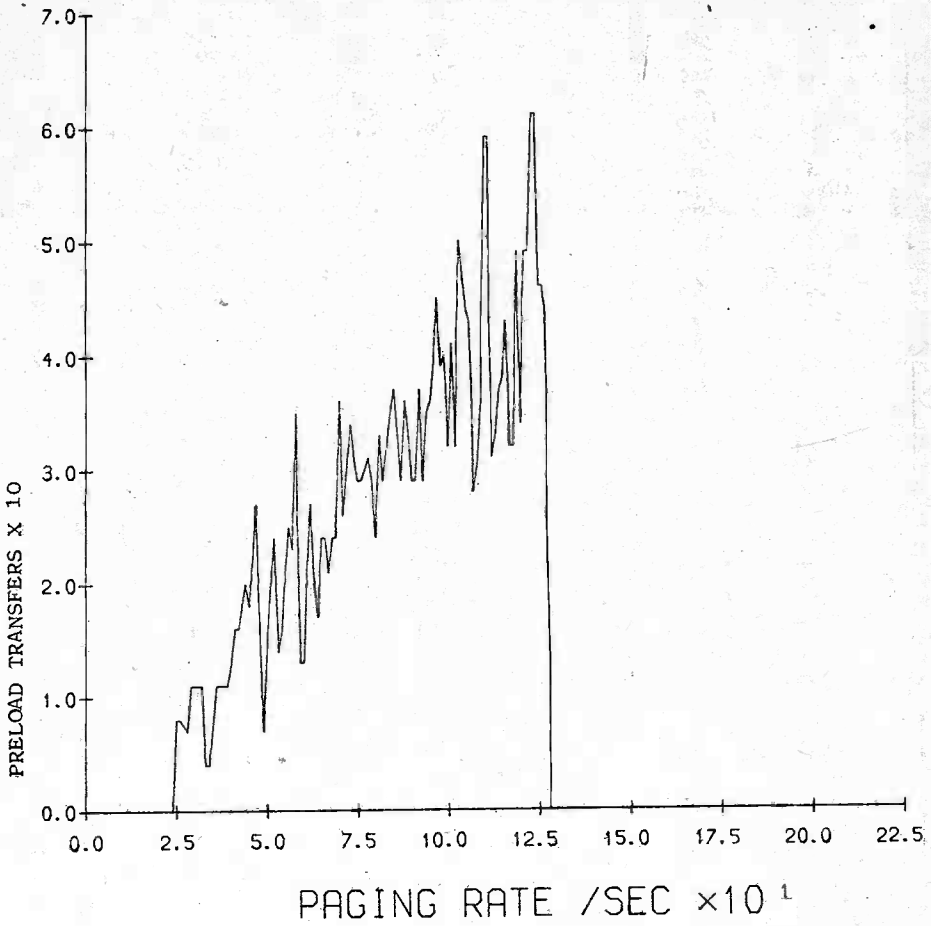


Figure 5.8 (e)

EXPERIMENT G

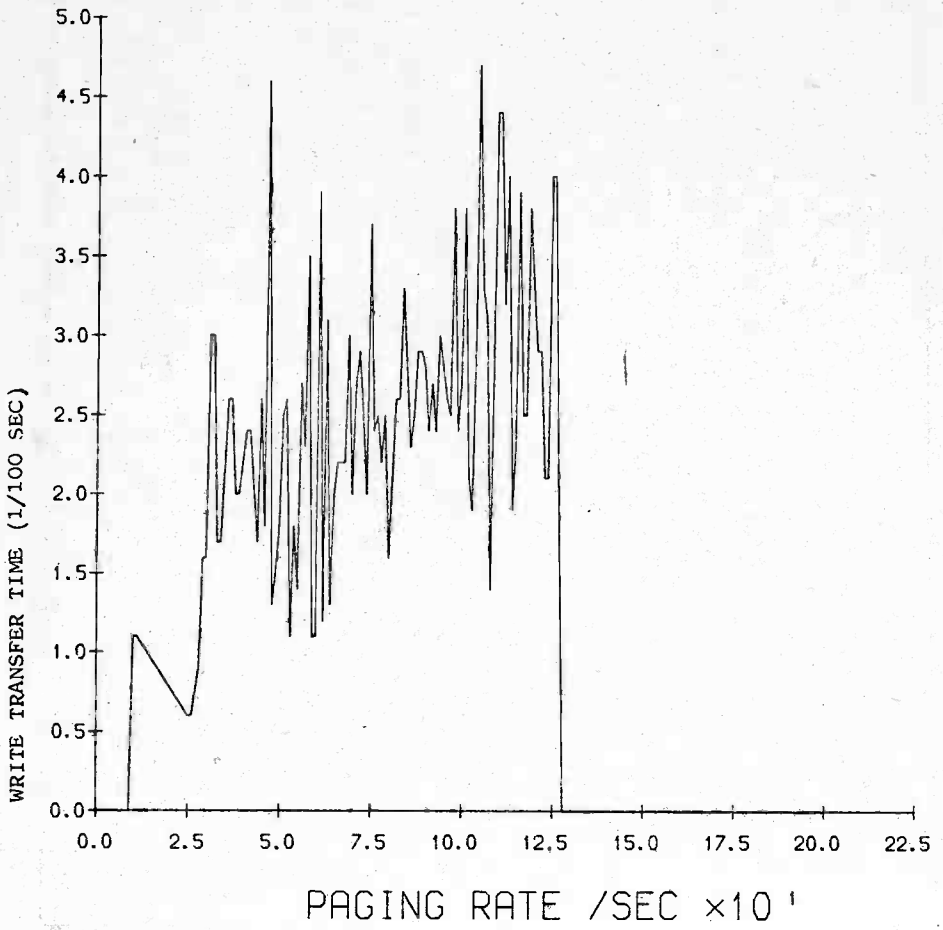


Figure 5.8 (f)

EXPERIMENT G

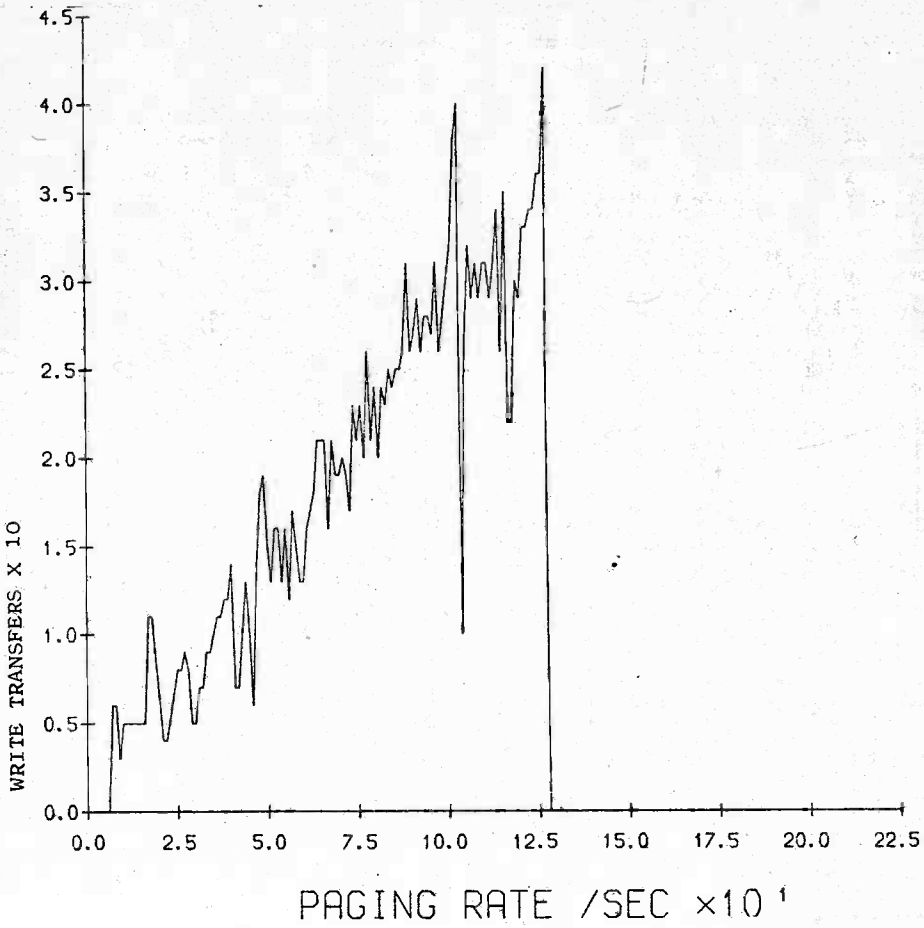


Figure 5.8 (g)

EXPERIMENT G

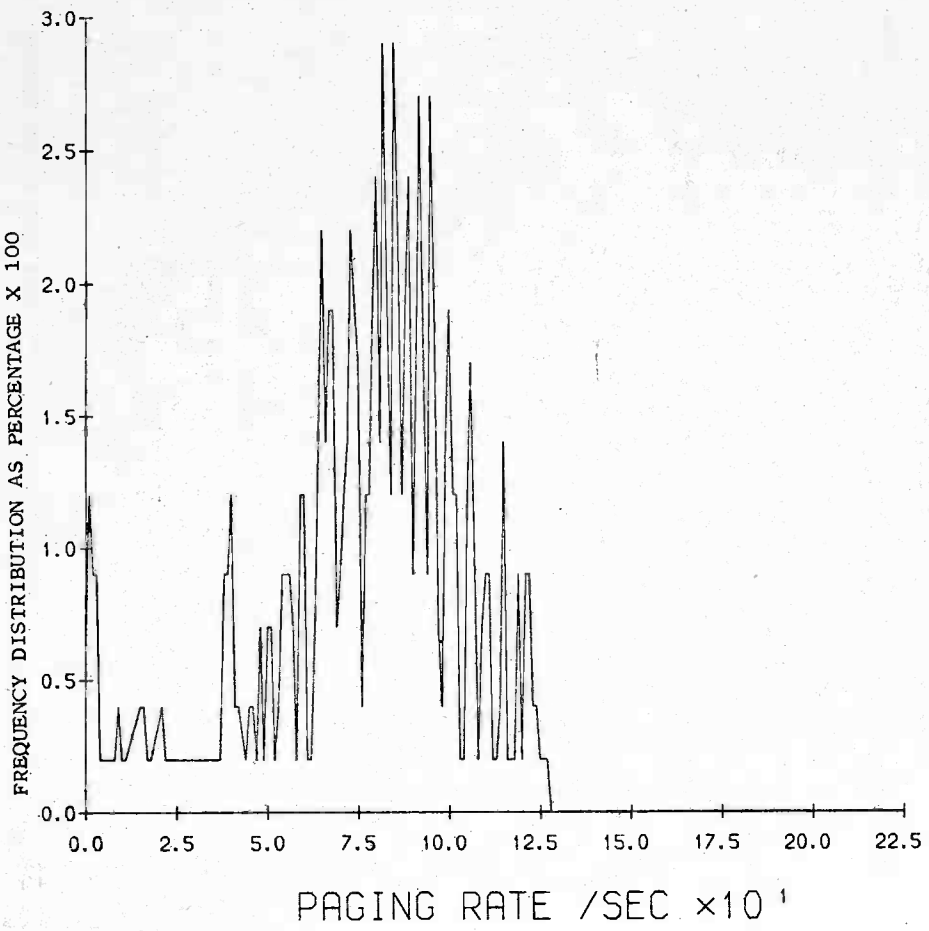


Figure 5.9 (a)

EXPERIMENT H

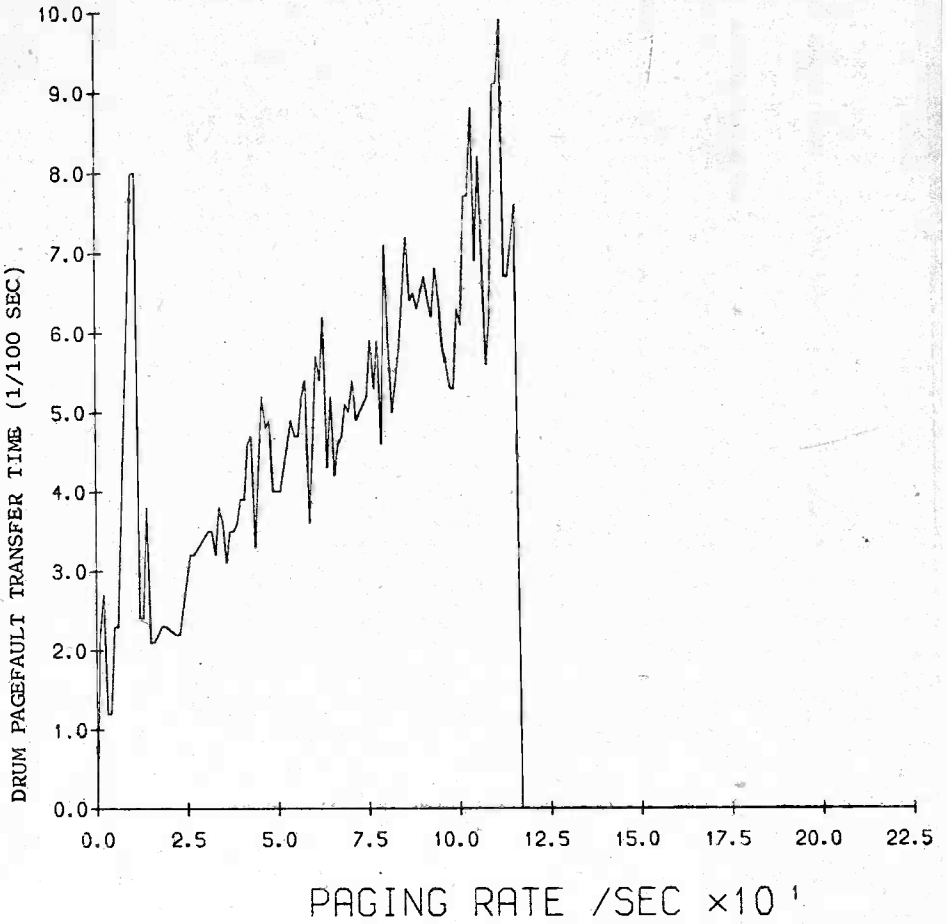


Figure 5.9 (b)

EXPERIMENT H

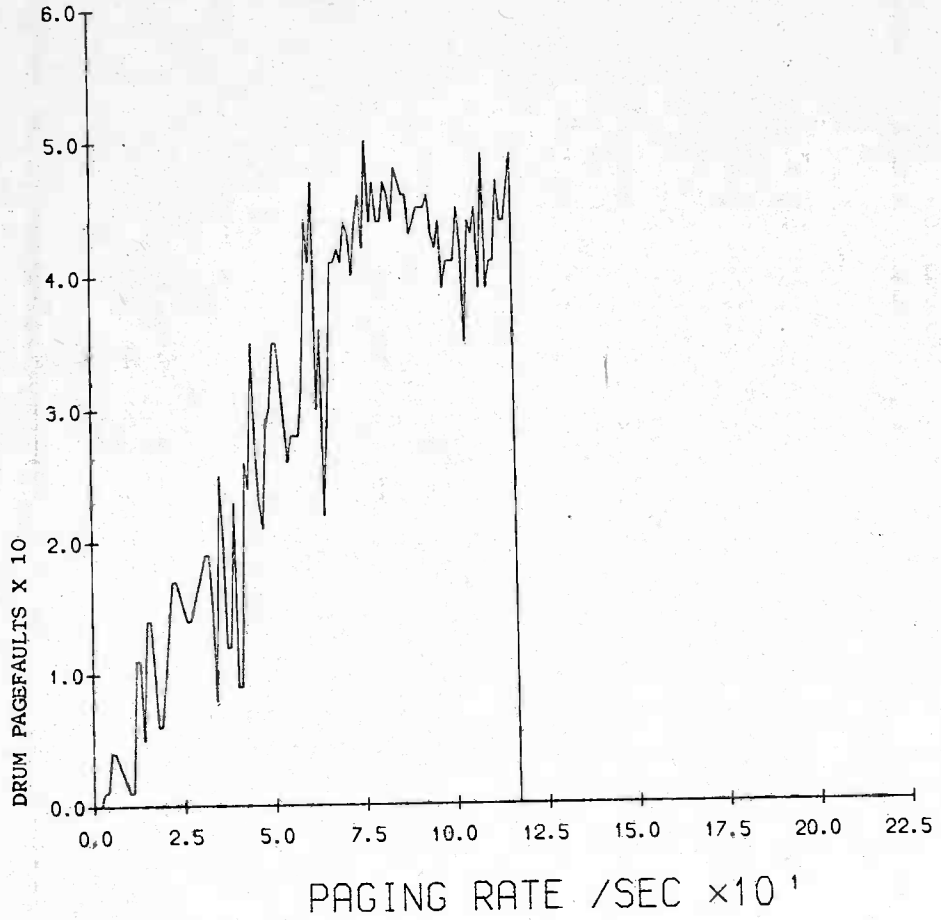


Figure 5.9 (c)

EXPERIMENT H

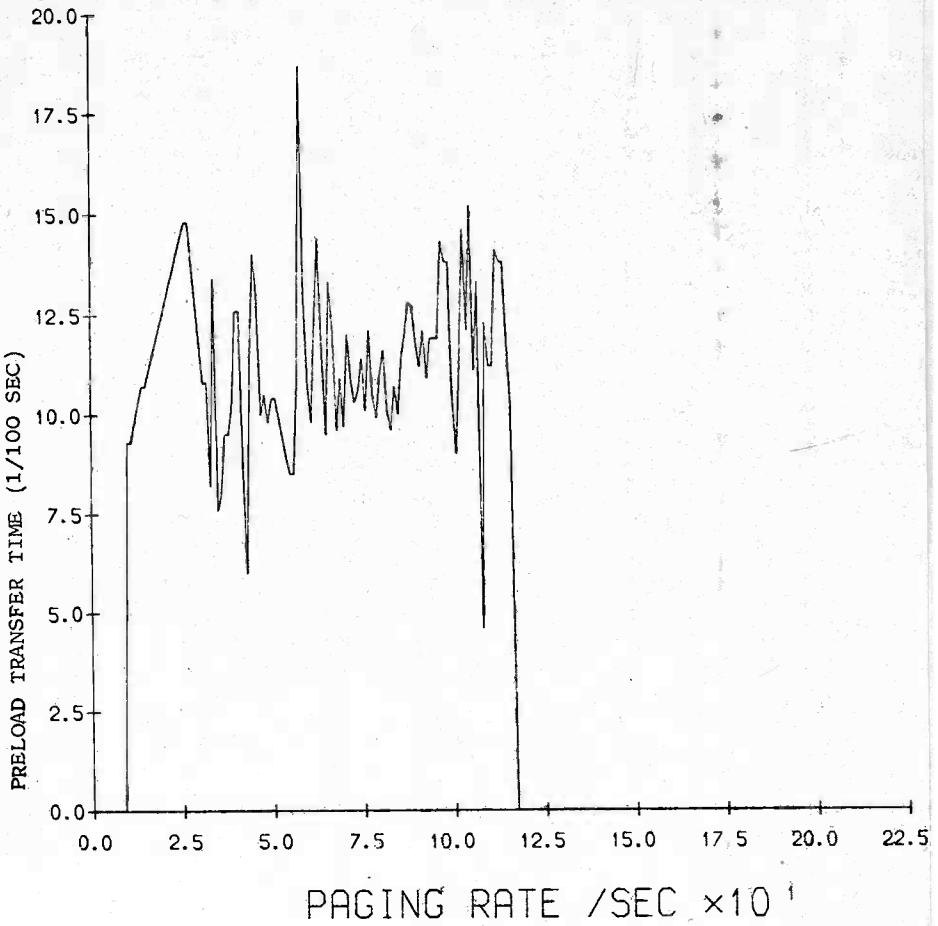


Figure 5.9 (d)

EXPERIMENT H

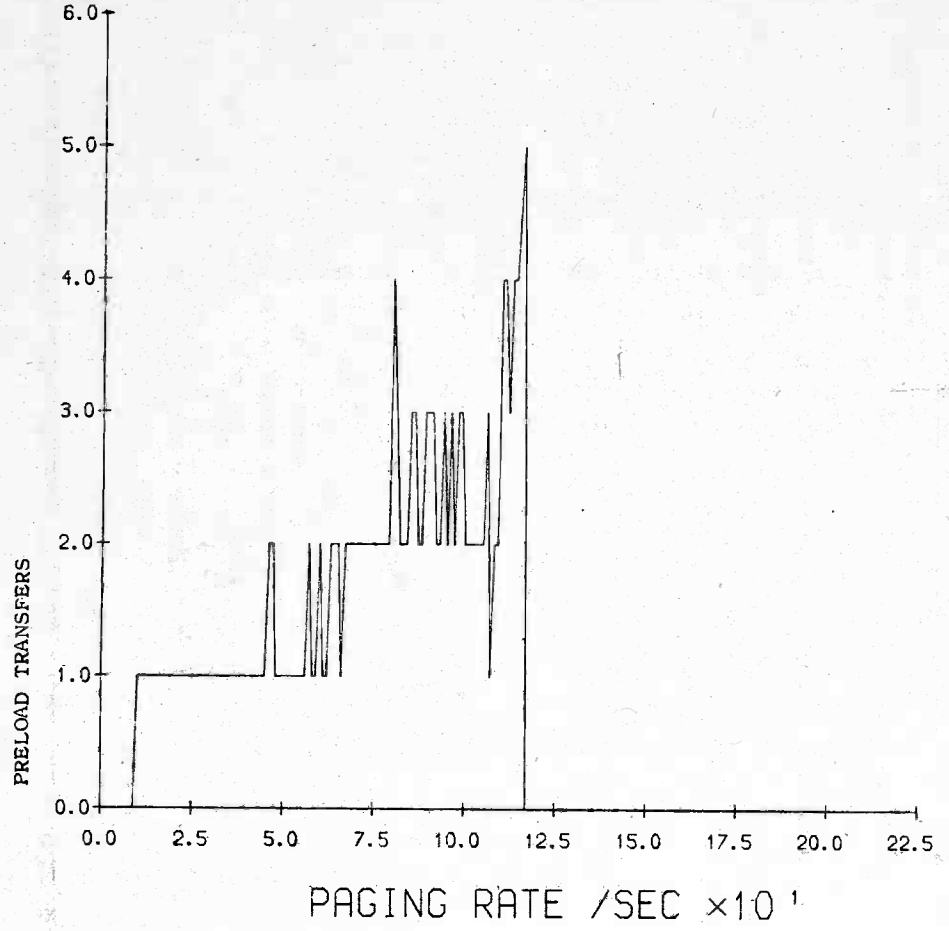


Figure 5.9 (e)

EXPERIMENT H

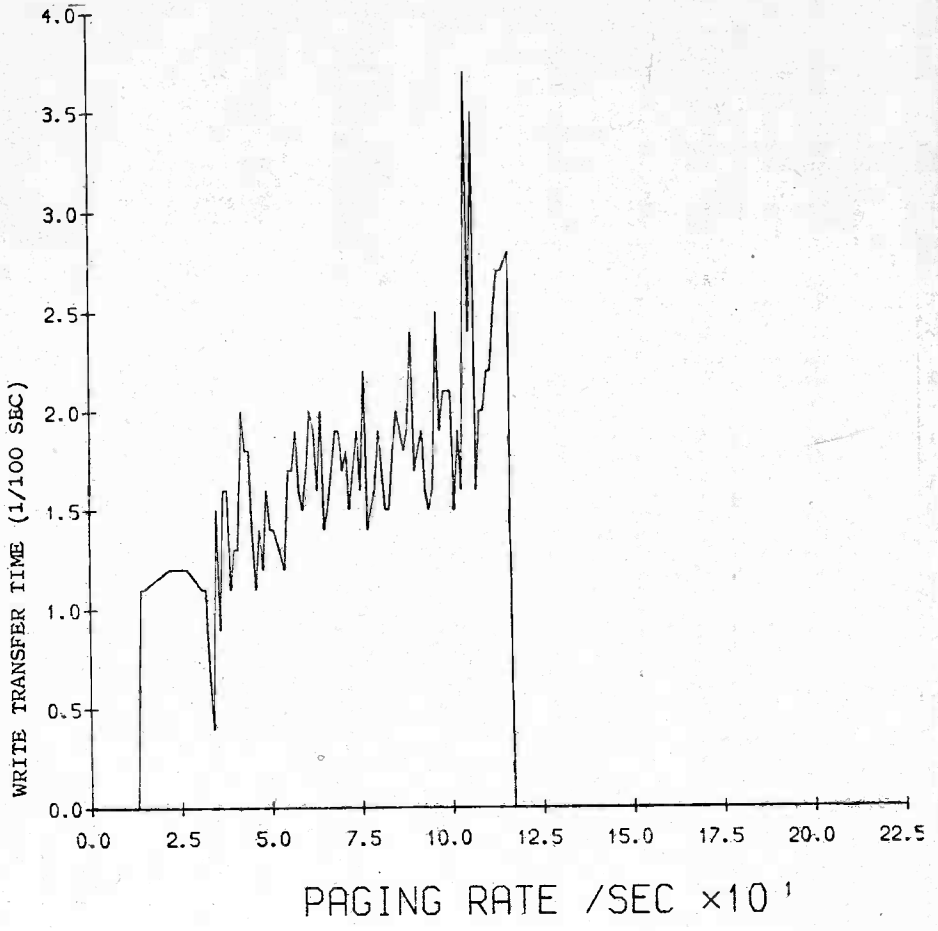


Figure 5.9 (f)

EXPERIMENT H

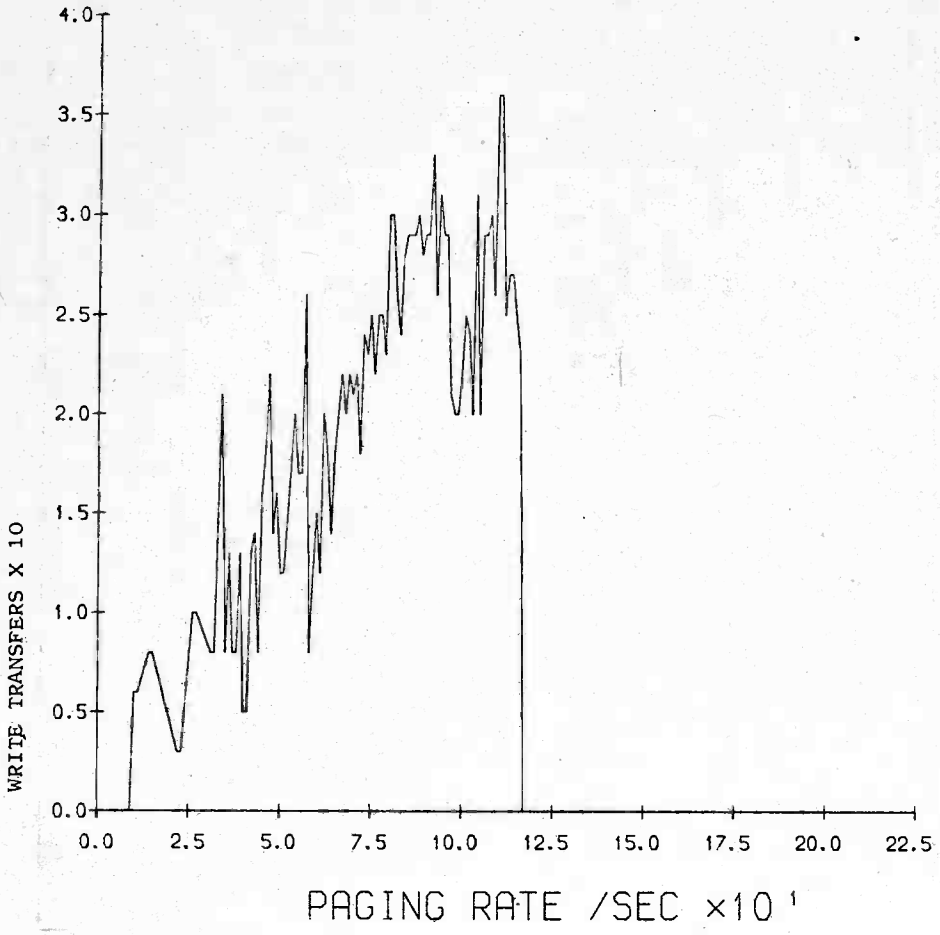


Figure 5.9 (g)

EXPERIMENT H

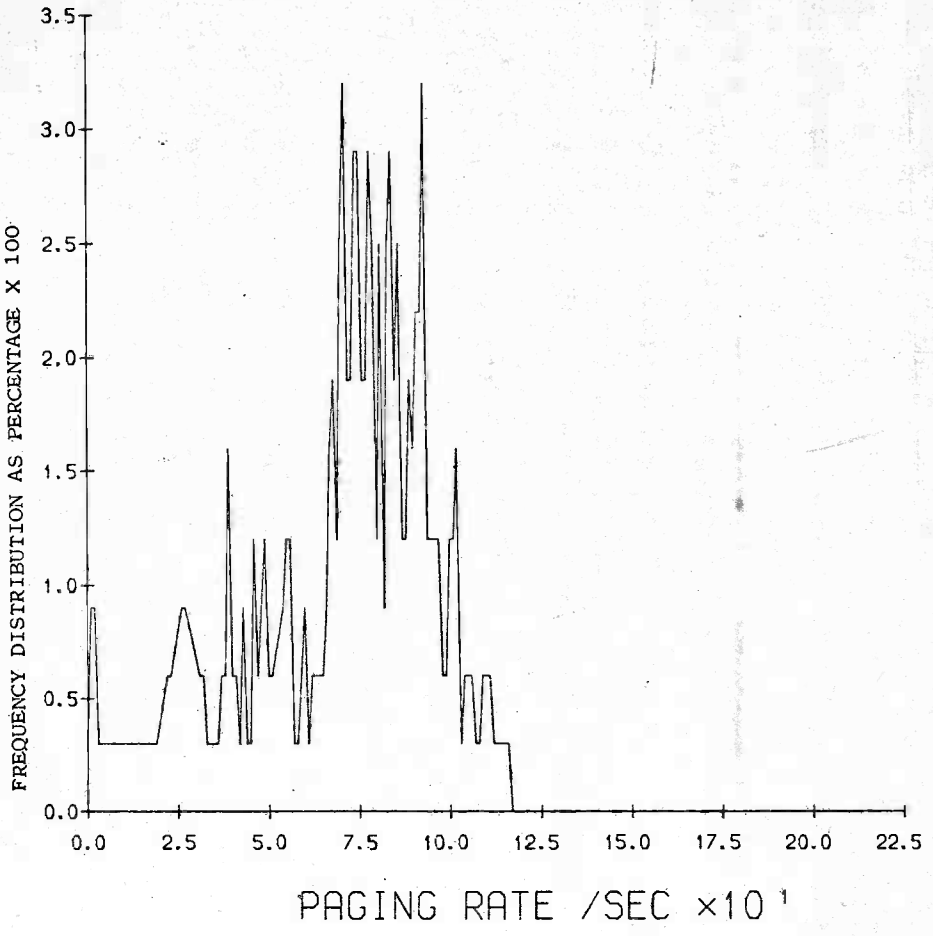


Figure 5.10 (a)

EXPERIMENT I

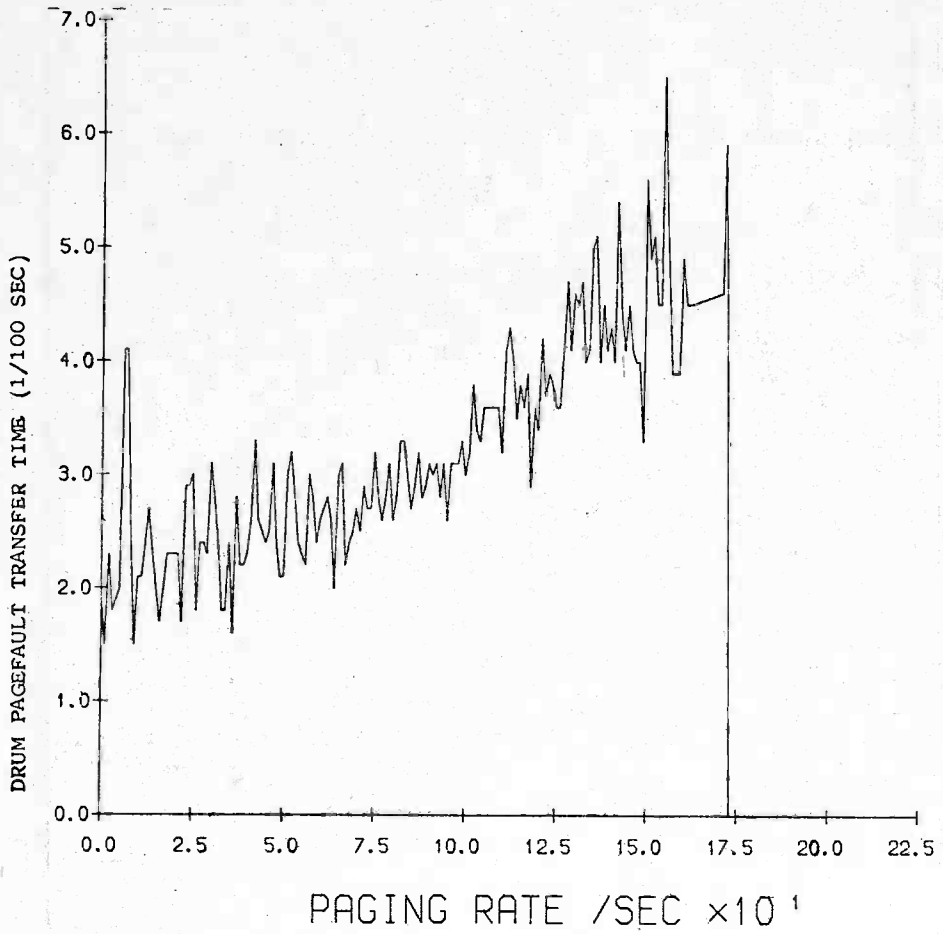


Figure 5.10 (b)

EXPERIMENT I

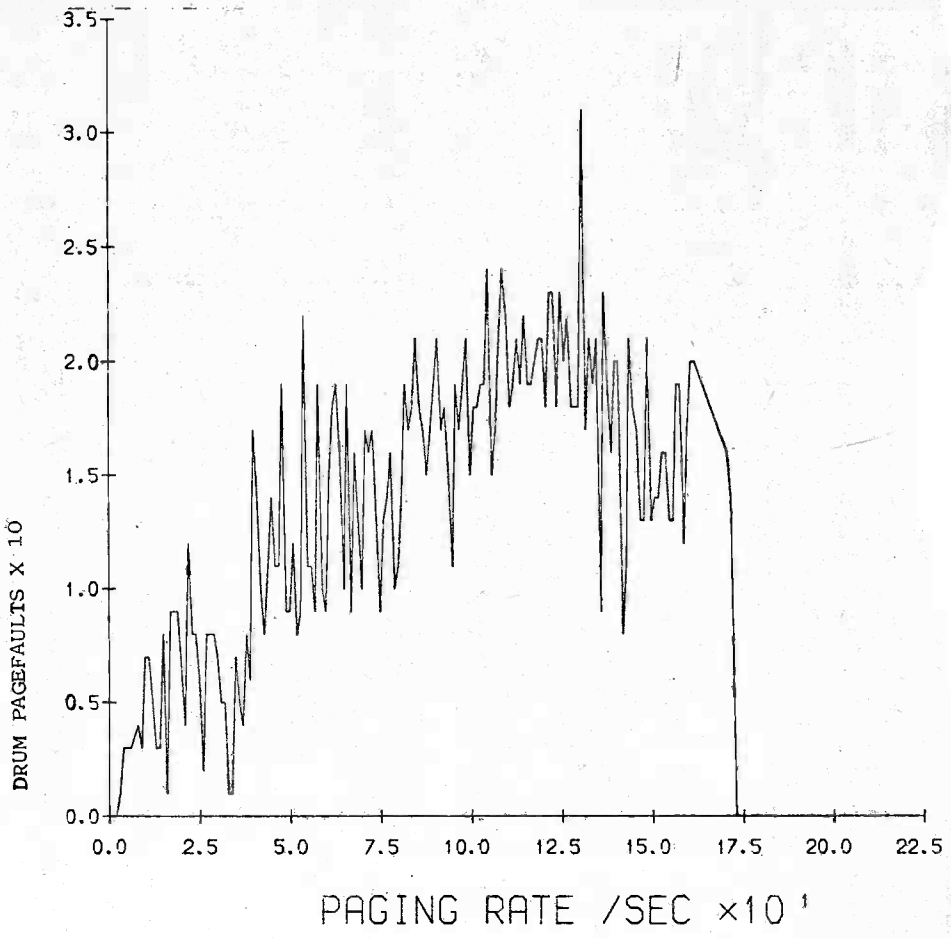


Figure 5.10 (c)

EXPERIMENT I

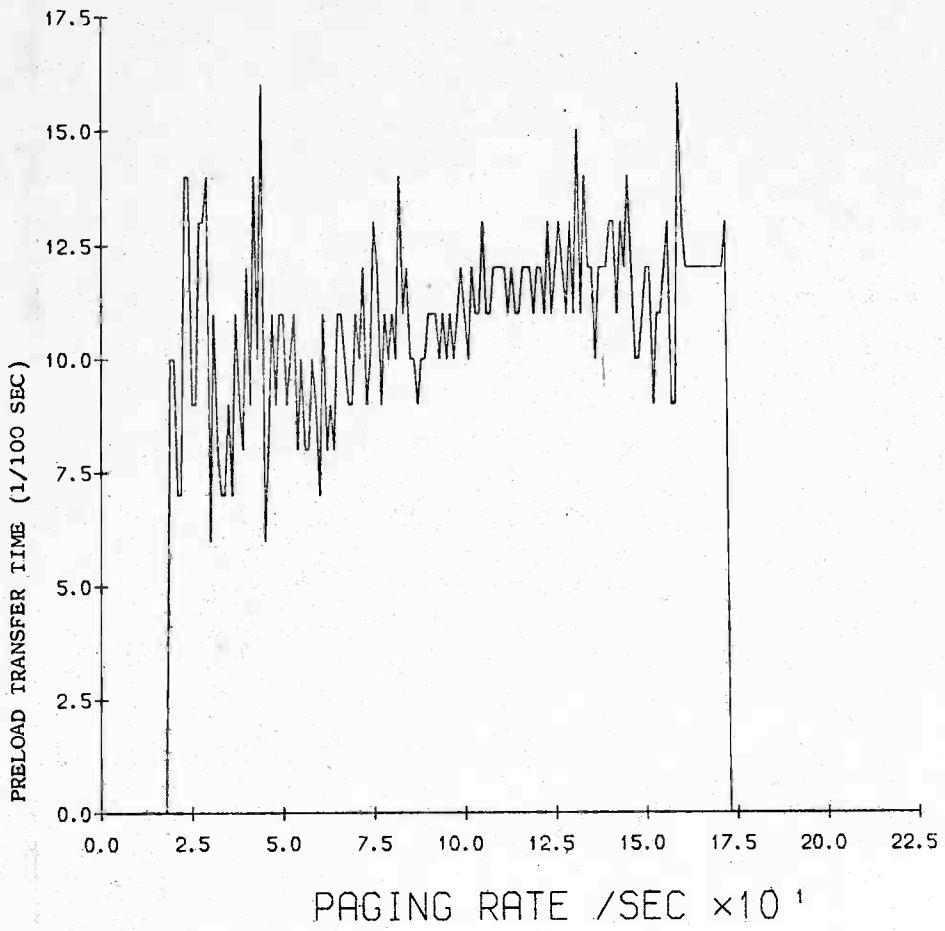


Figure 5.10 (d)

EXPERIMENT I

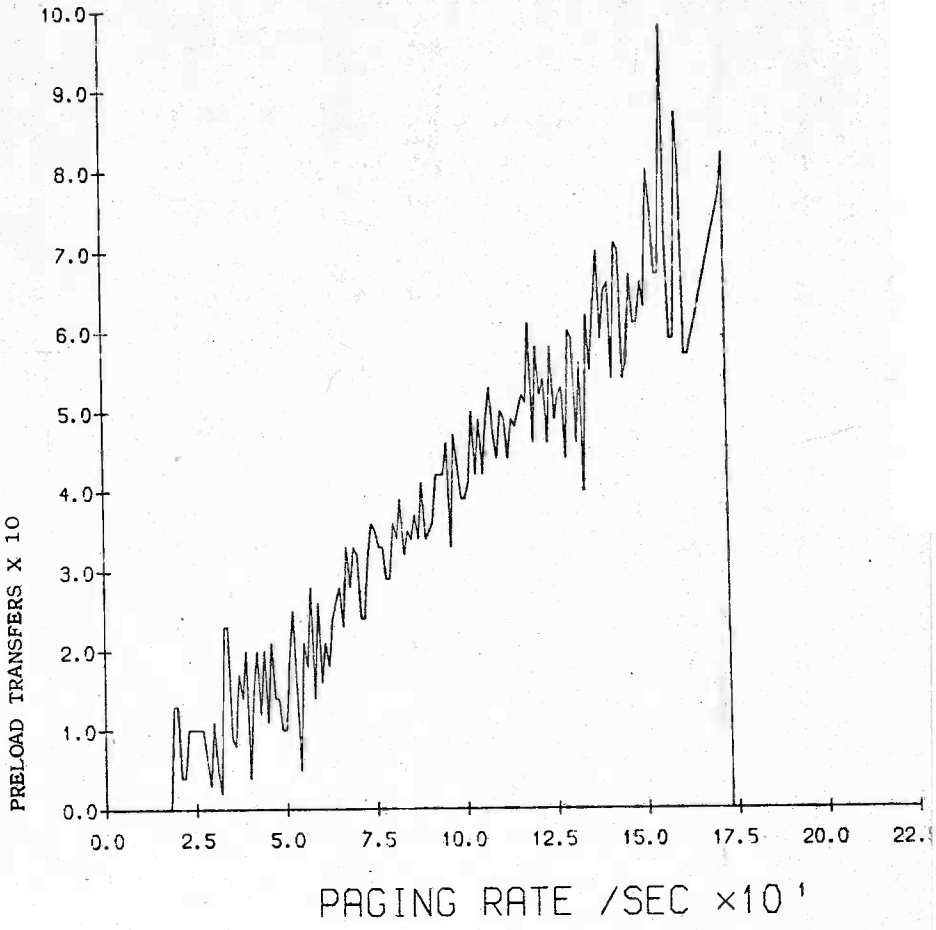


Figure 5.10 (e)

EXPERIMENT I

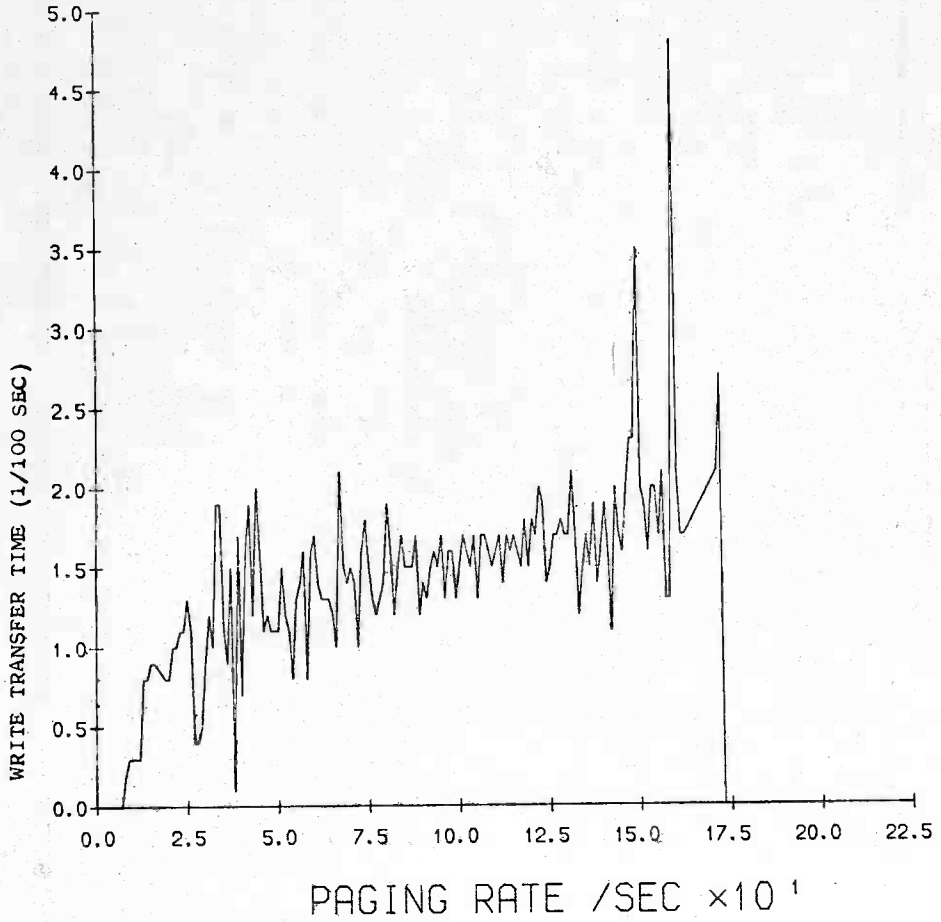


Figure 5.10 (f)

EXPERIMENT I

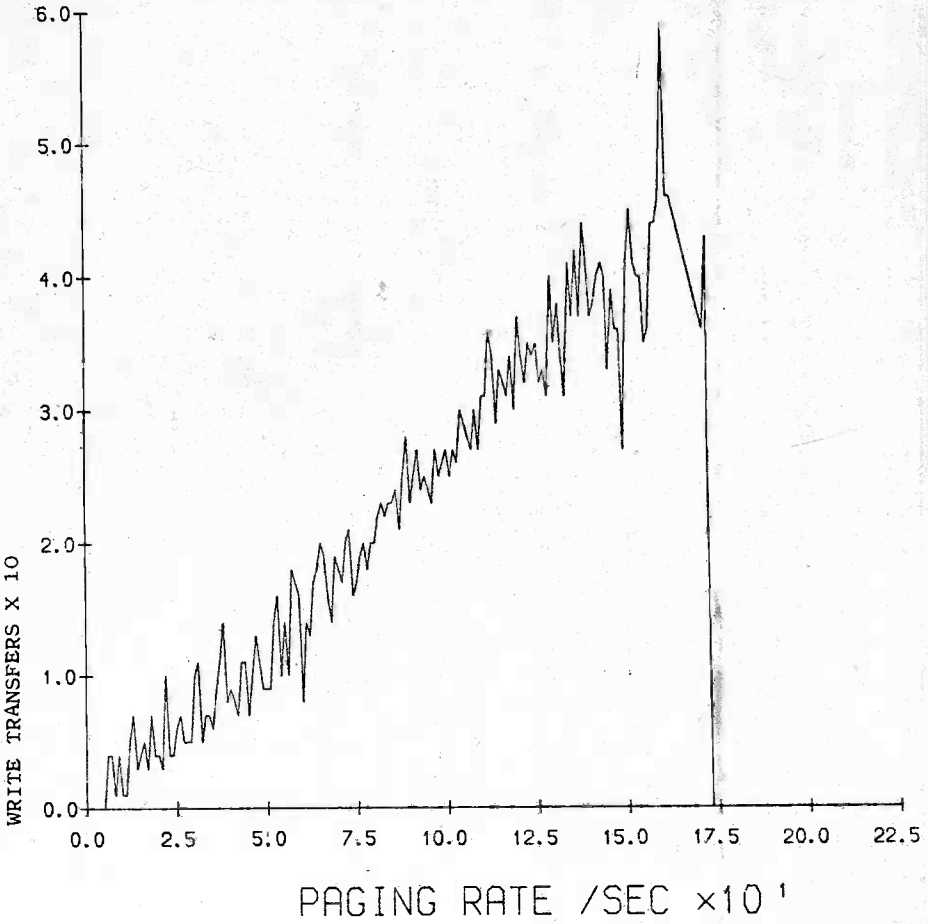


Figure 5.10 (g)

EXPERIMENT I

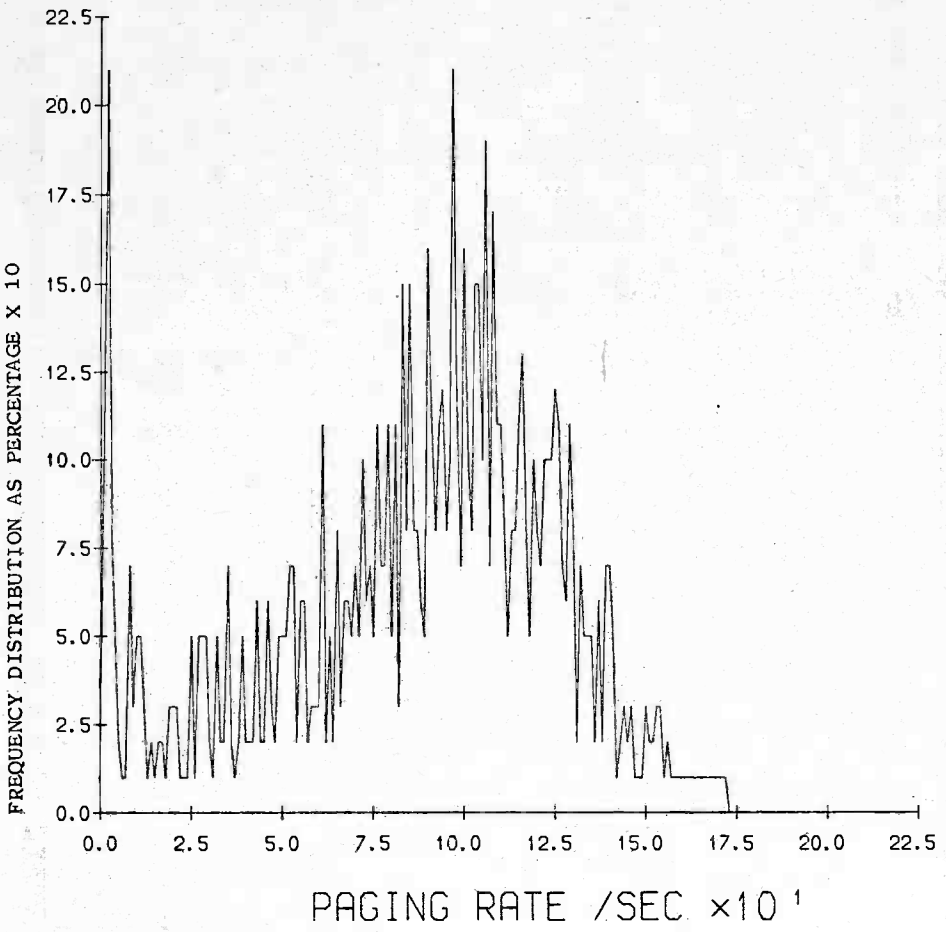


Figure 5.11 (a)

EXPERIMENT J

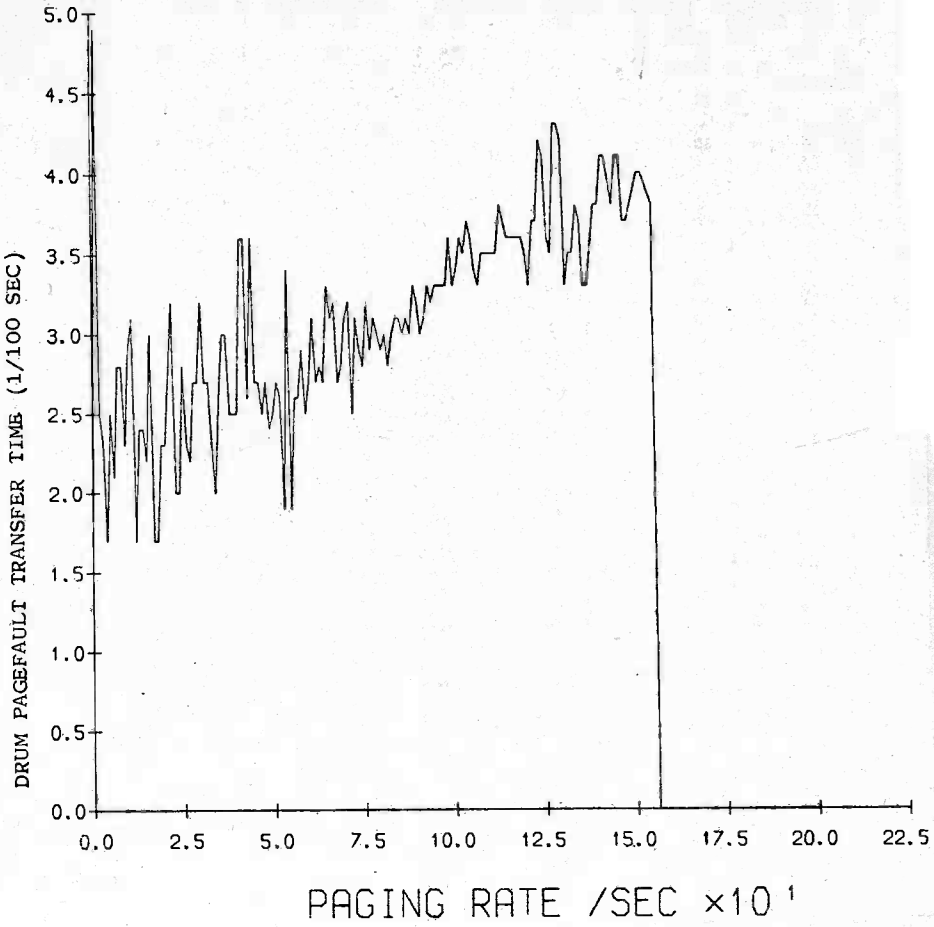


Figure 5.11 (b)

EXPERIMENT J

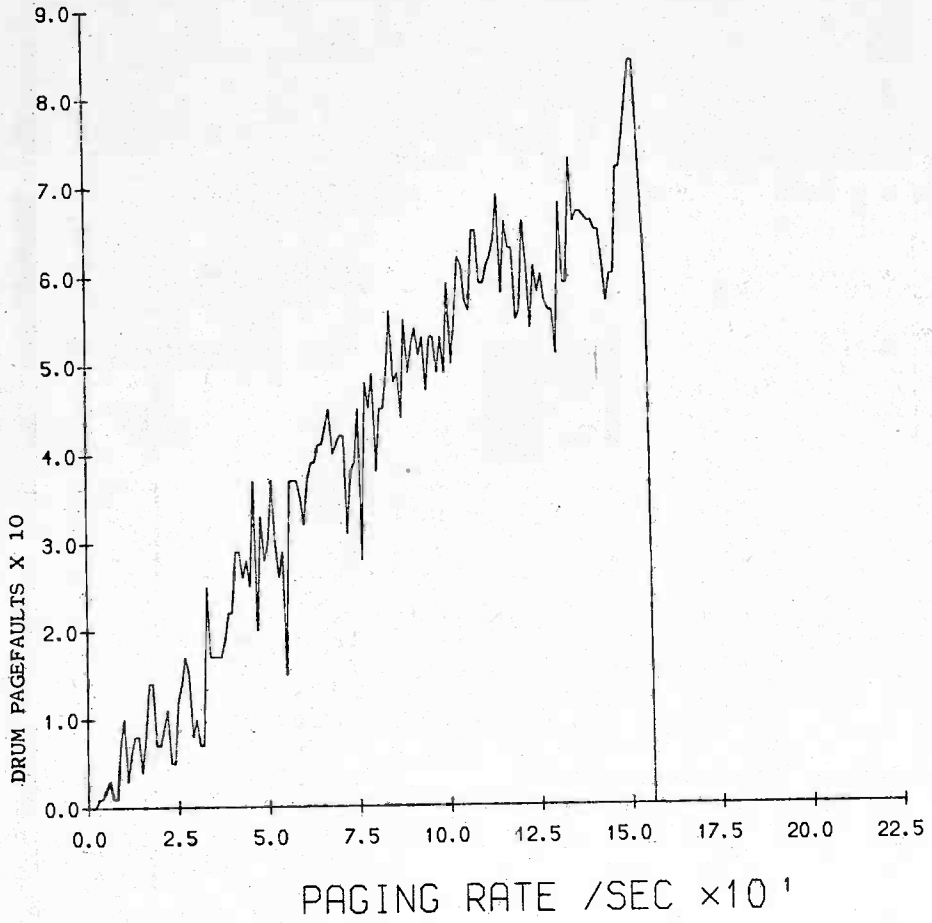


Figure 5.11 (c)

EXPERIMENT J

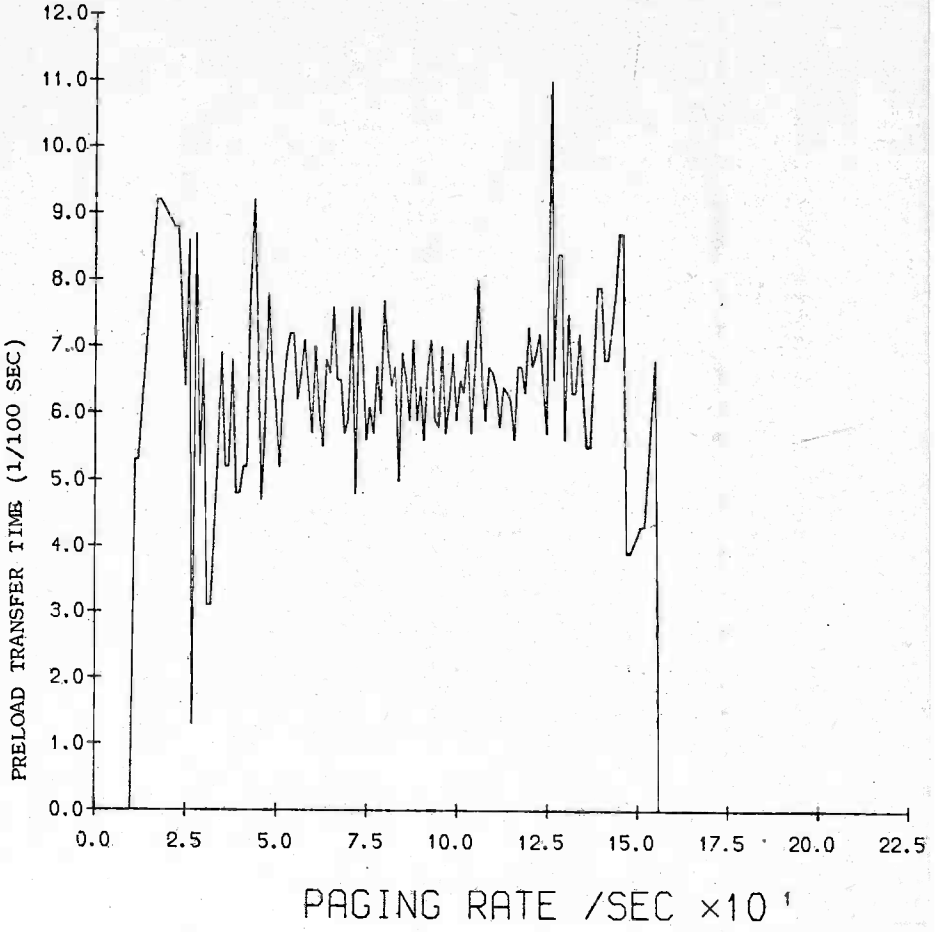


Figure 5.11 (d)

EXPERIMENT J

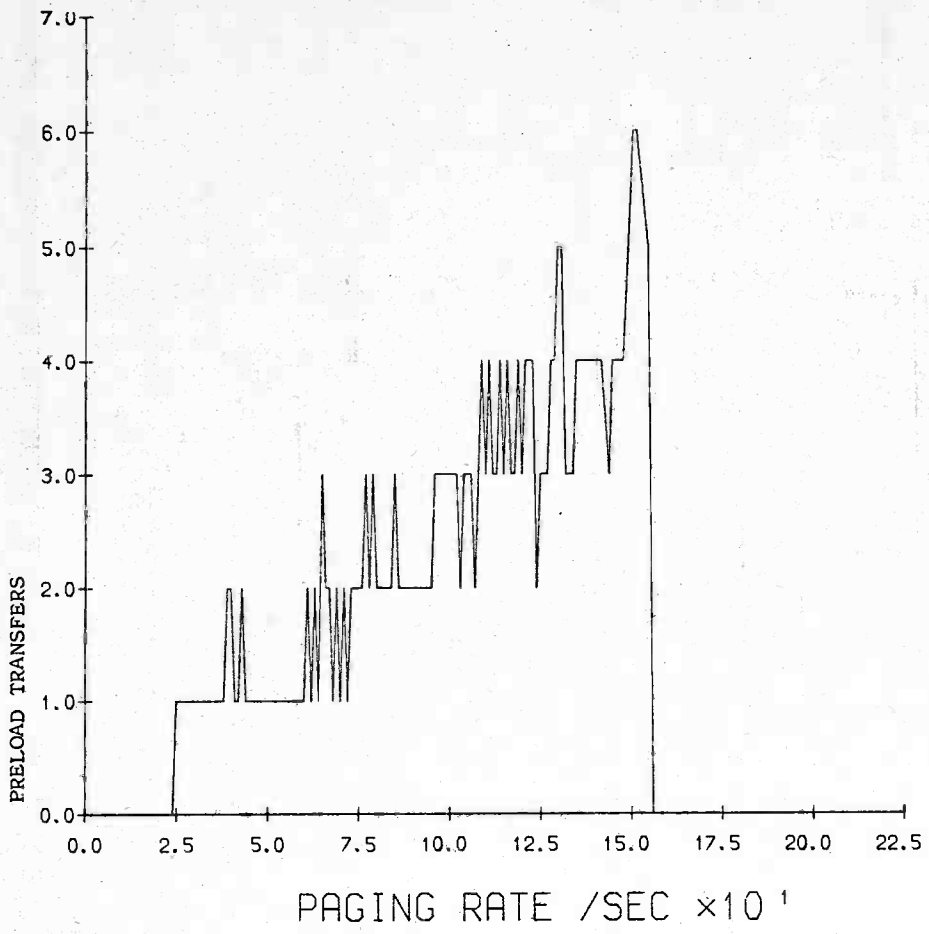


Figure 5.11 (e)

EXPERIMENT J

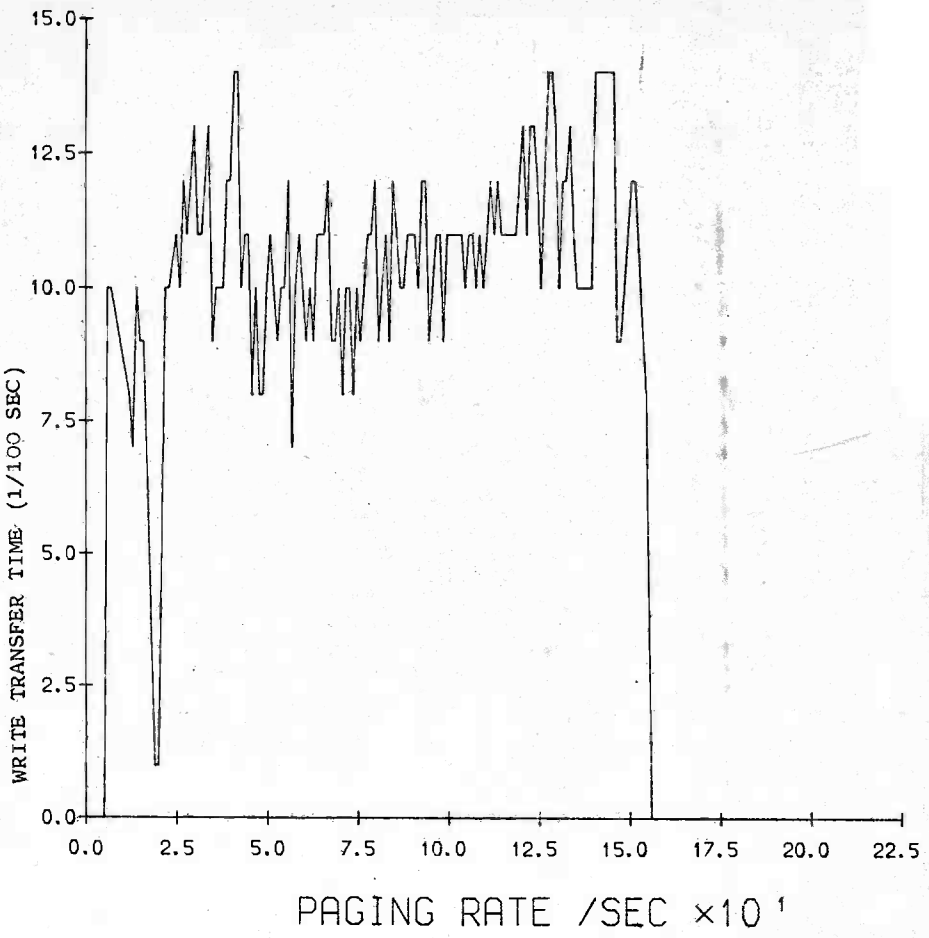


Figure 5.11 (f)

EXPERIMENT J

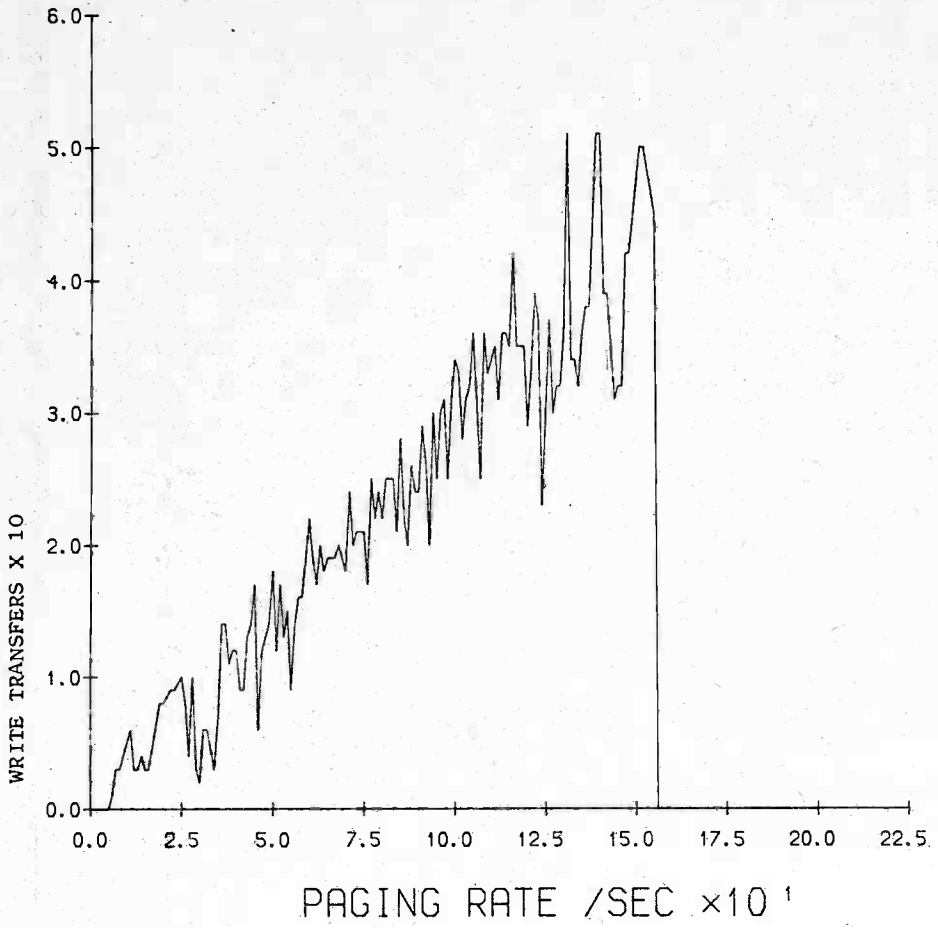


Figure 5.11 (g)

EXPERIMENT J

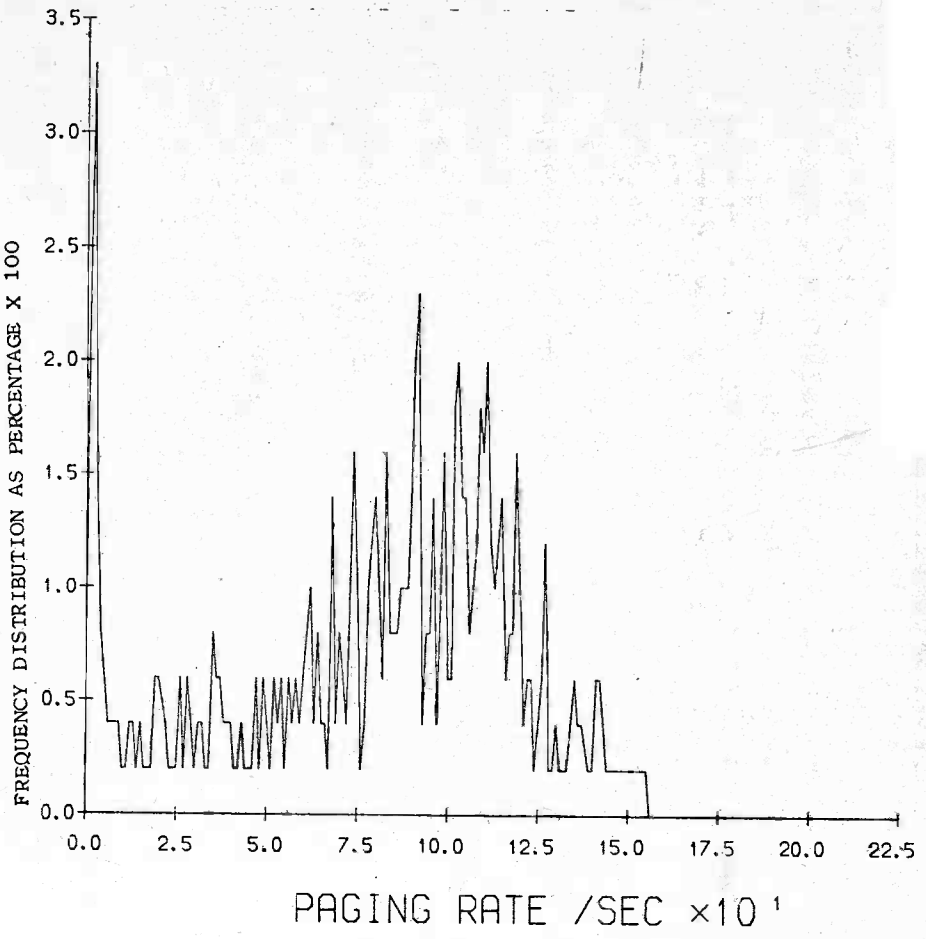


Figure 5.12 (a)

EXPERIMENT K

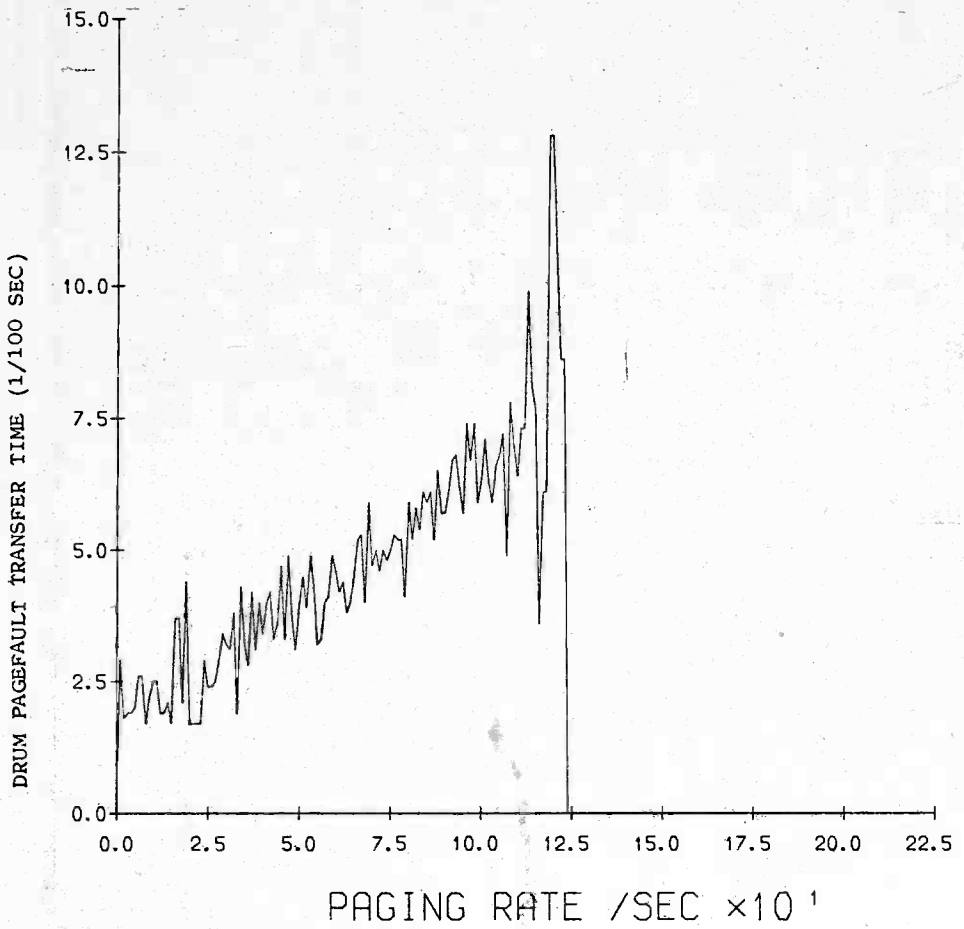


Figure 5.12 (b)

EXPERIMENT K

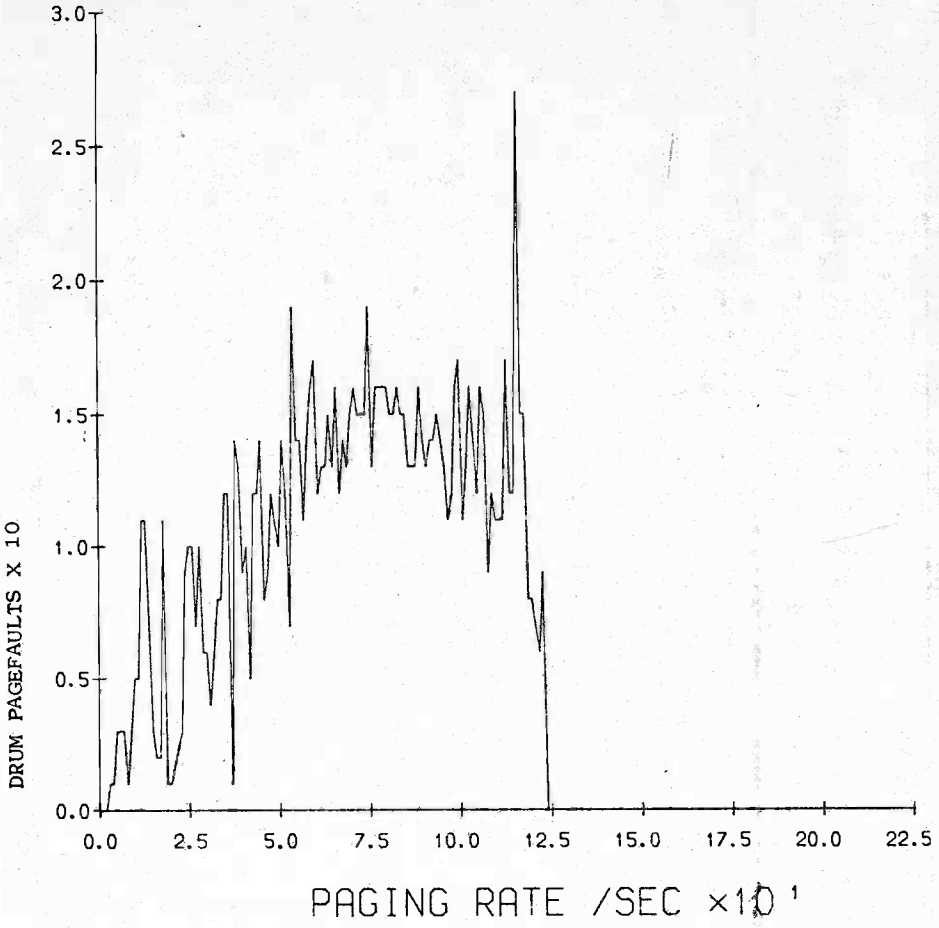


Figure 5.12 (c)

EXPERIMENT K

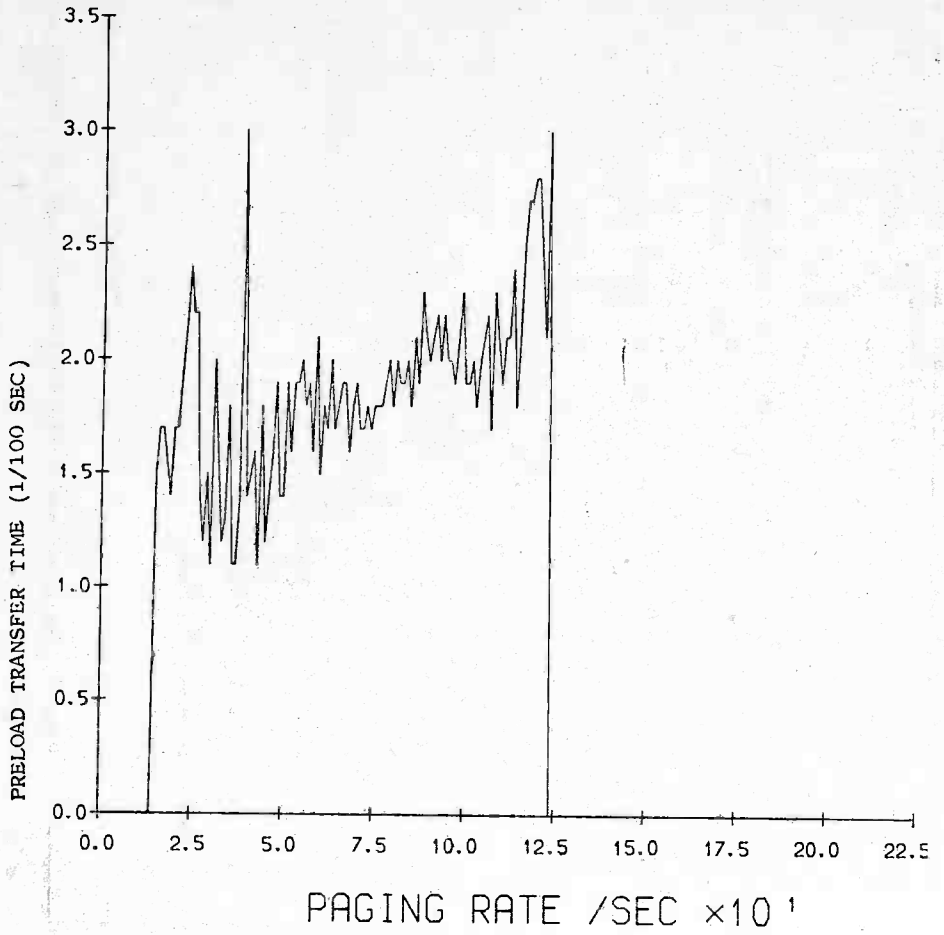


Figure 5.12 (d)

EXPERIMENT K

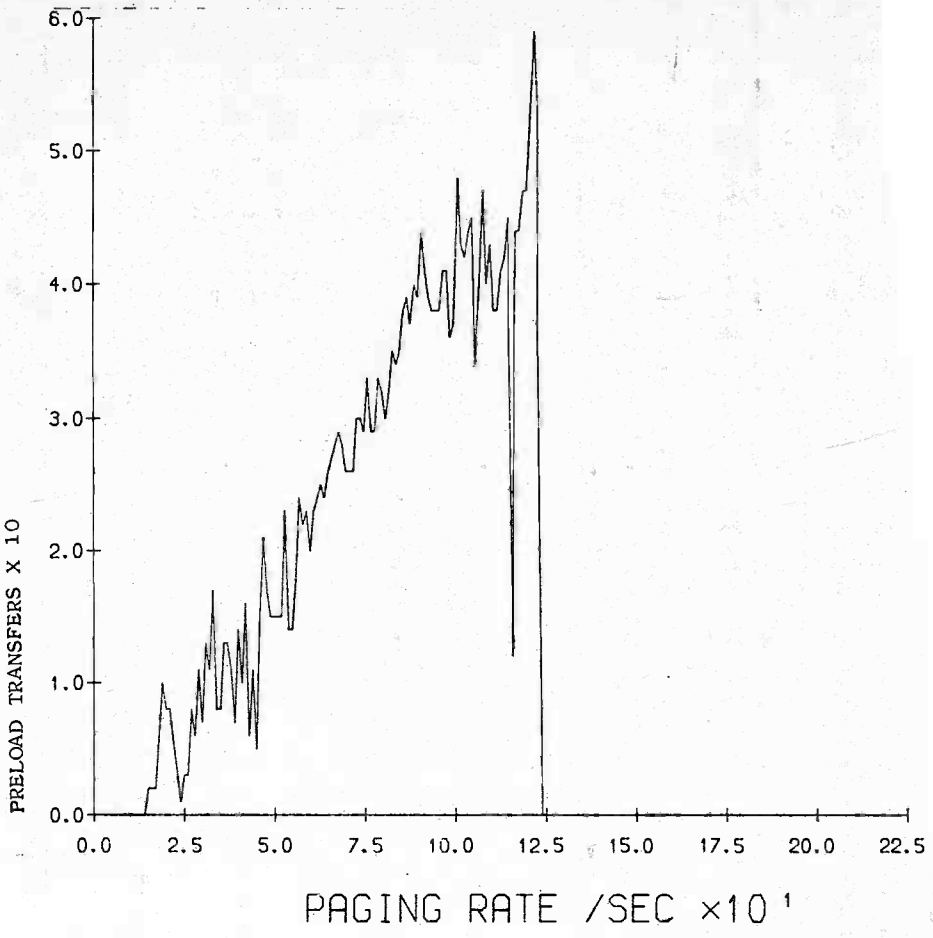


Figure 5.12 (e)

EXPERIMENT K

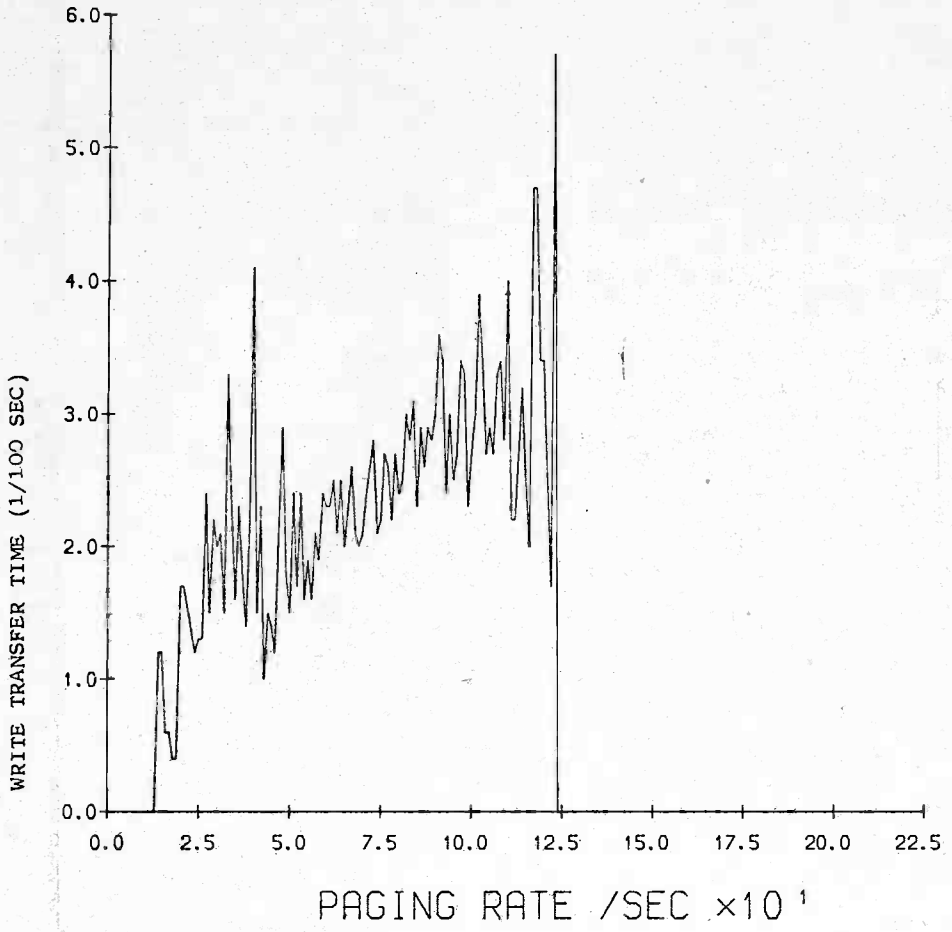


Figure 5.12 (f)

EXPERIMENT K

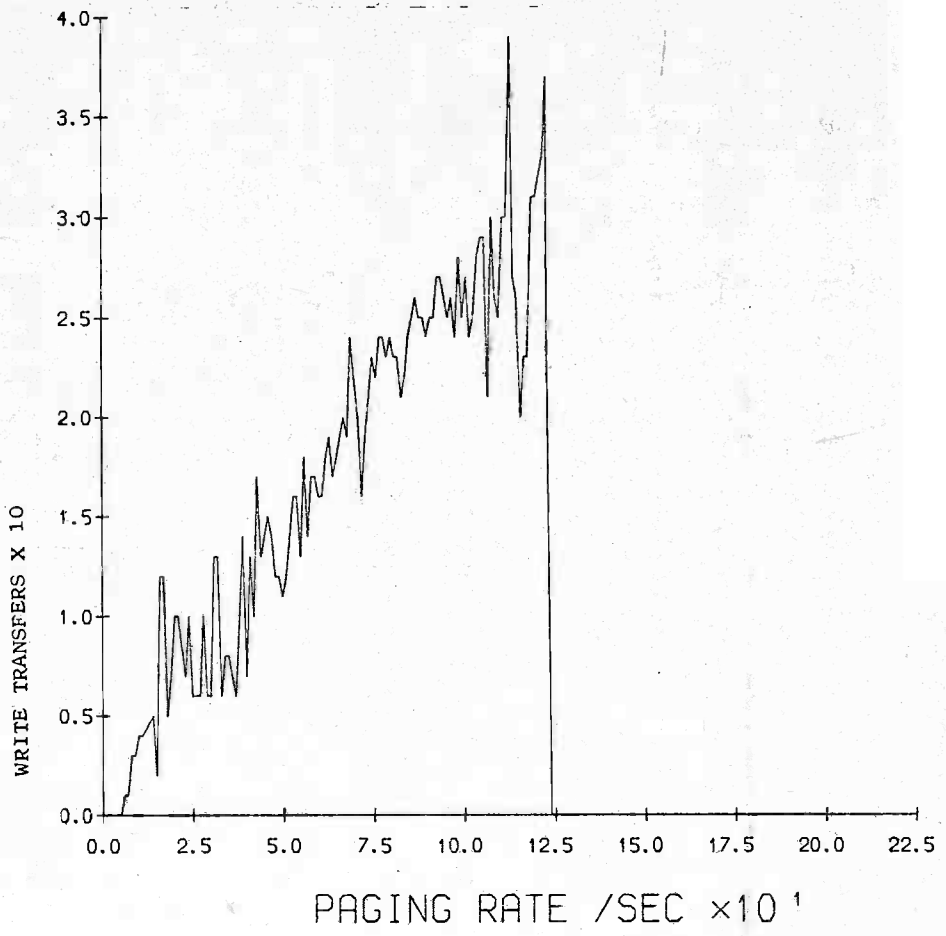


Figure 5.12 (g)

EXPERIMENT K

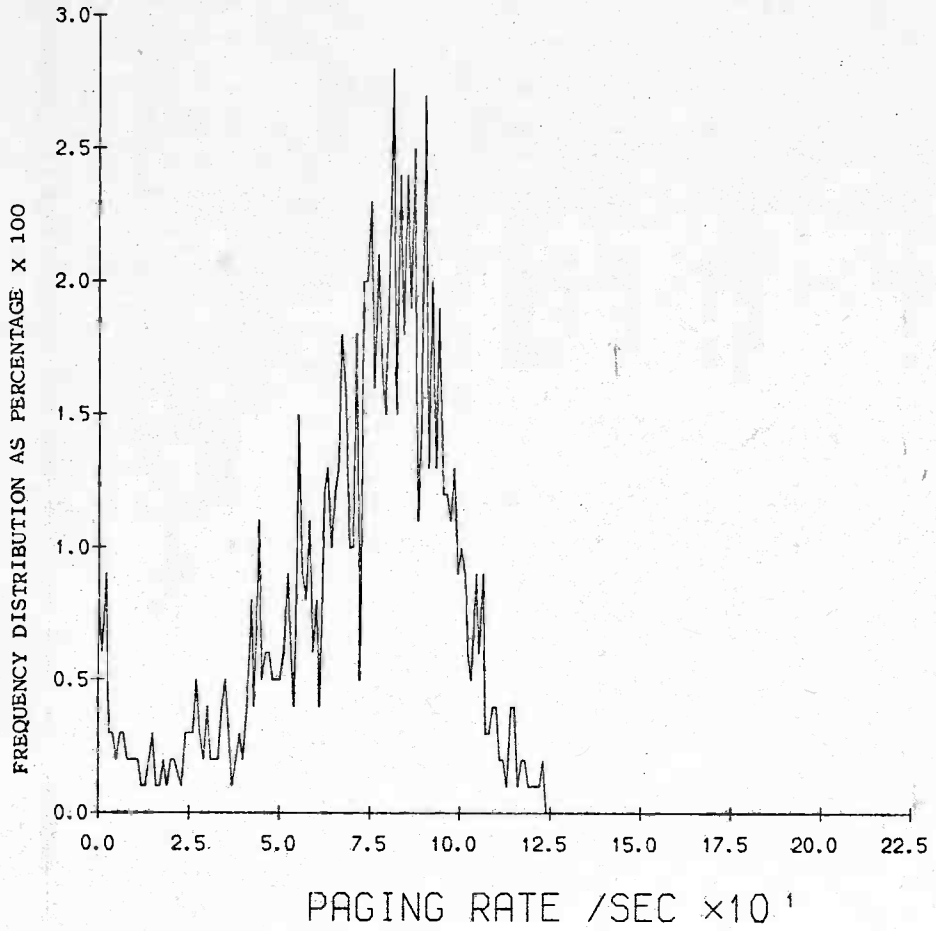


Figure 5.13 (a)

EXPERIMENT L

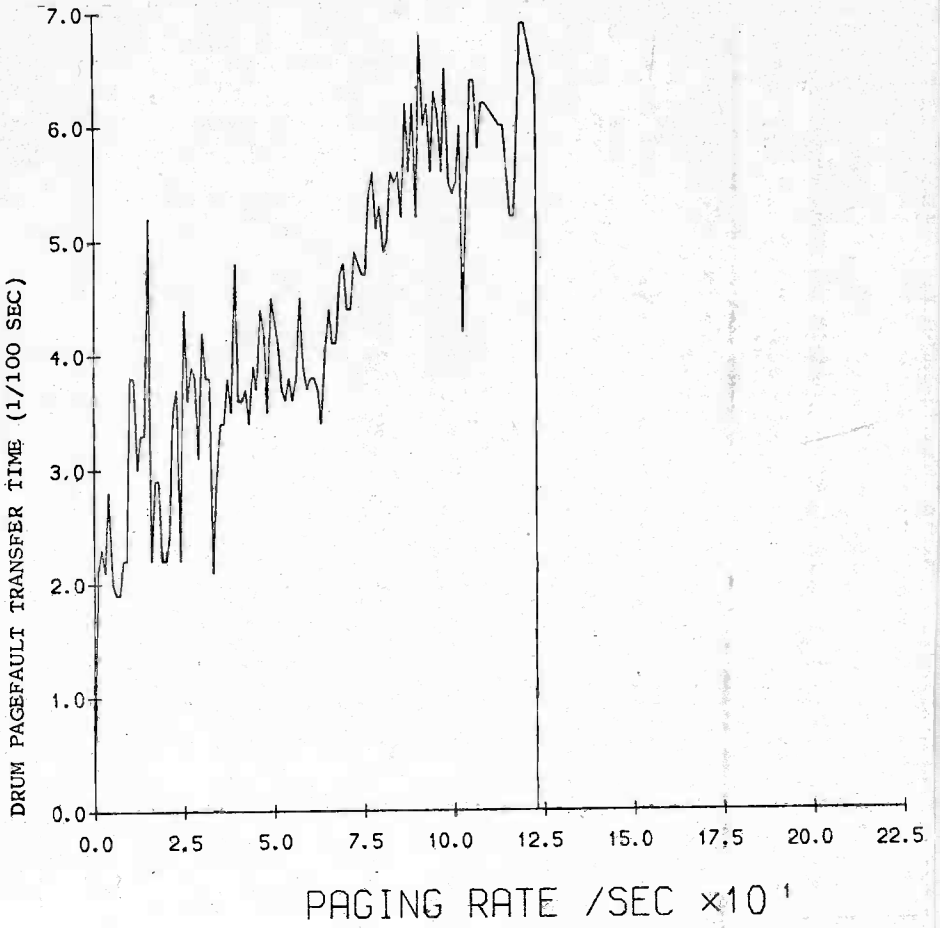


Figure 5.13 (b)

EXPERIMENT L

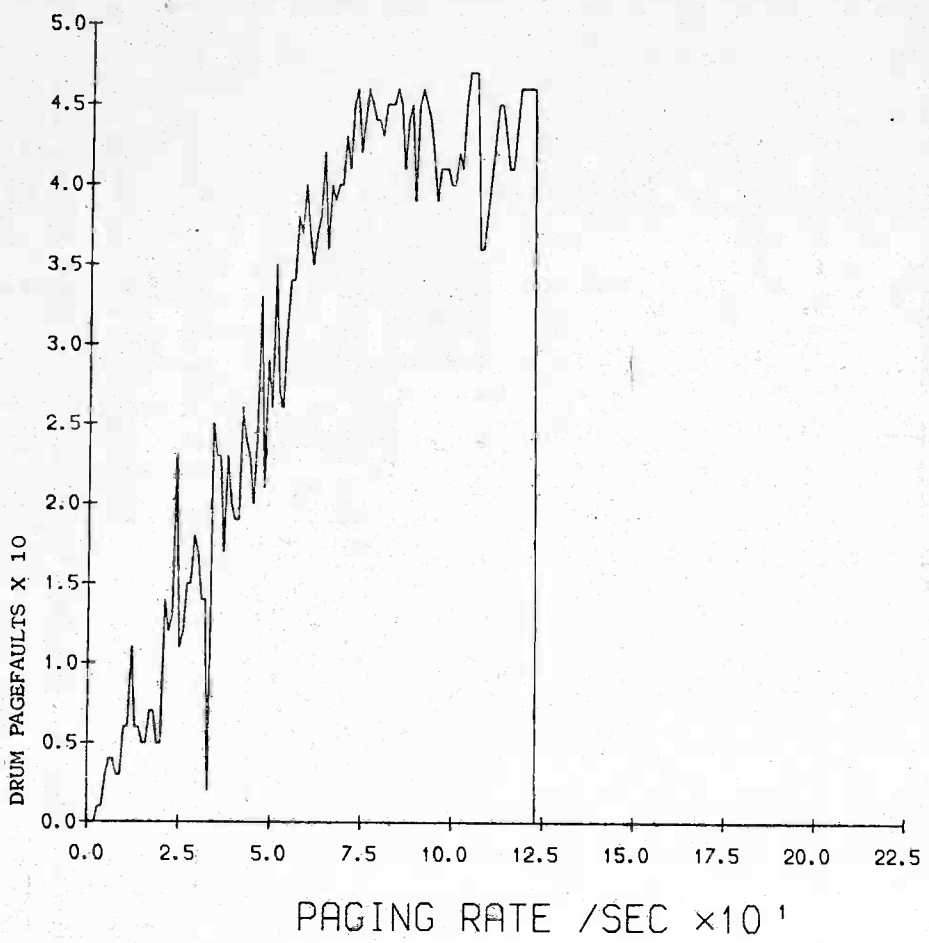


Figure 5.13 (c)

EXPERIMENT L

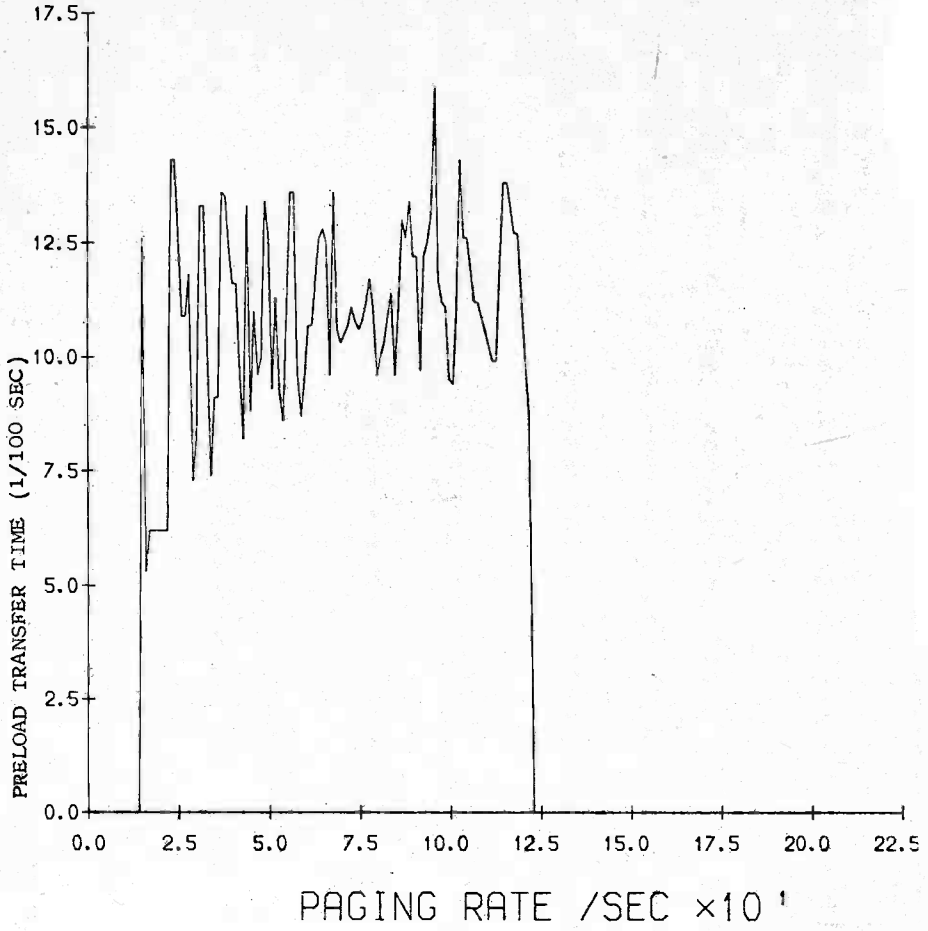


Figure 5.13 (d)

EXPERIMENT L

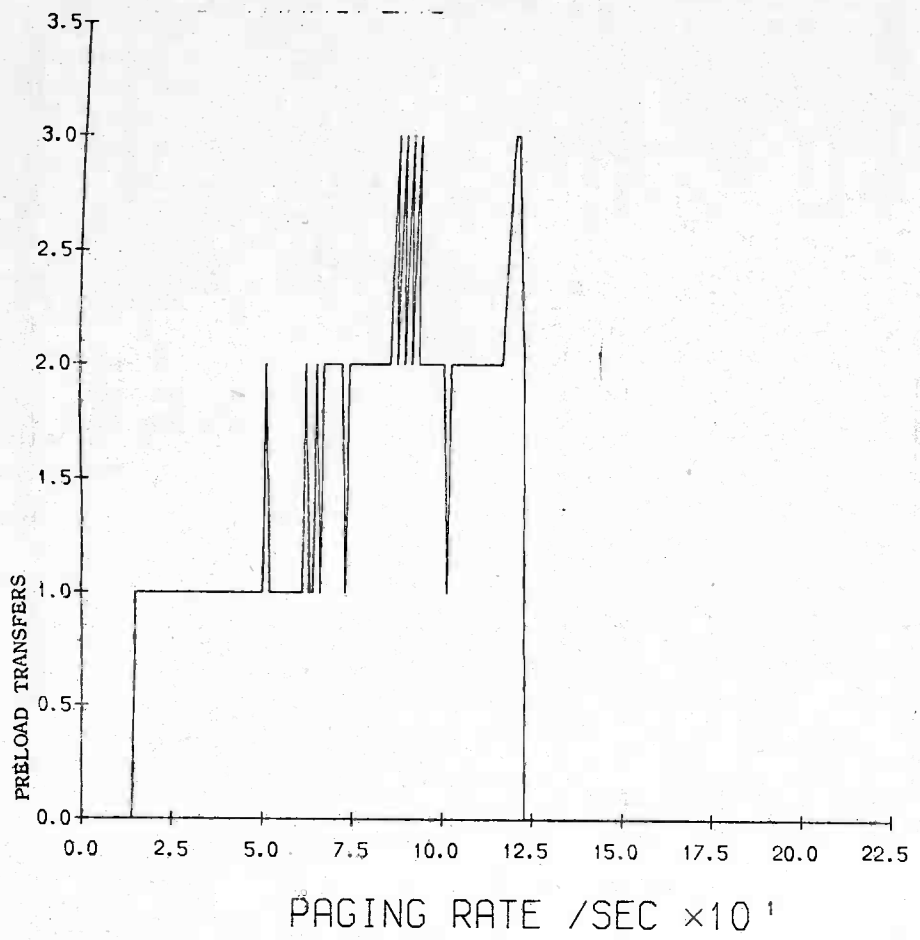


Figure 5.13 (e)

EXPERIMENT L

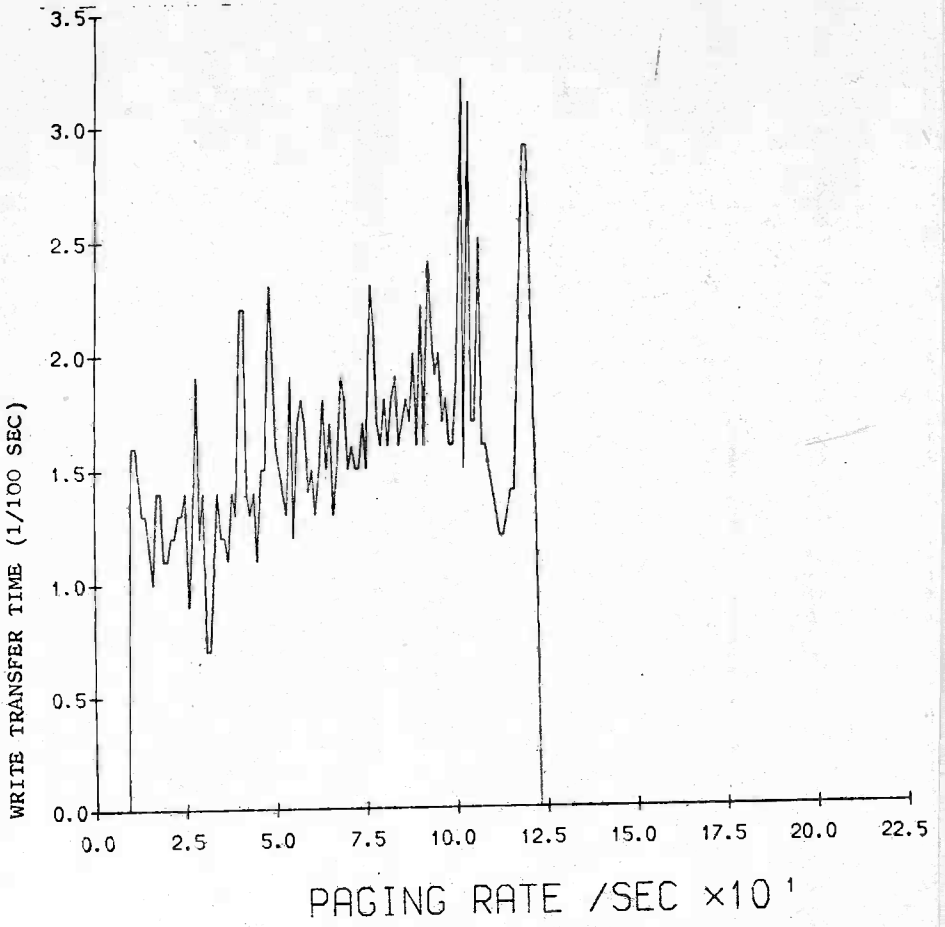


Figure 5.13 (f)

EXPERIMENT L

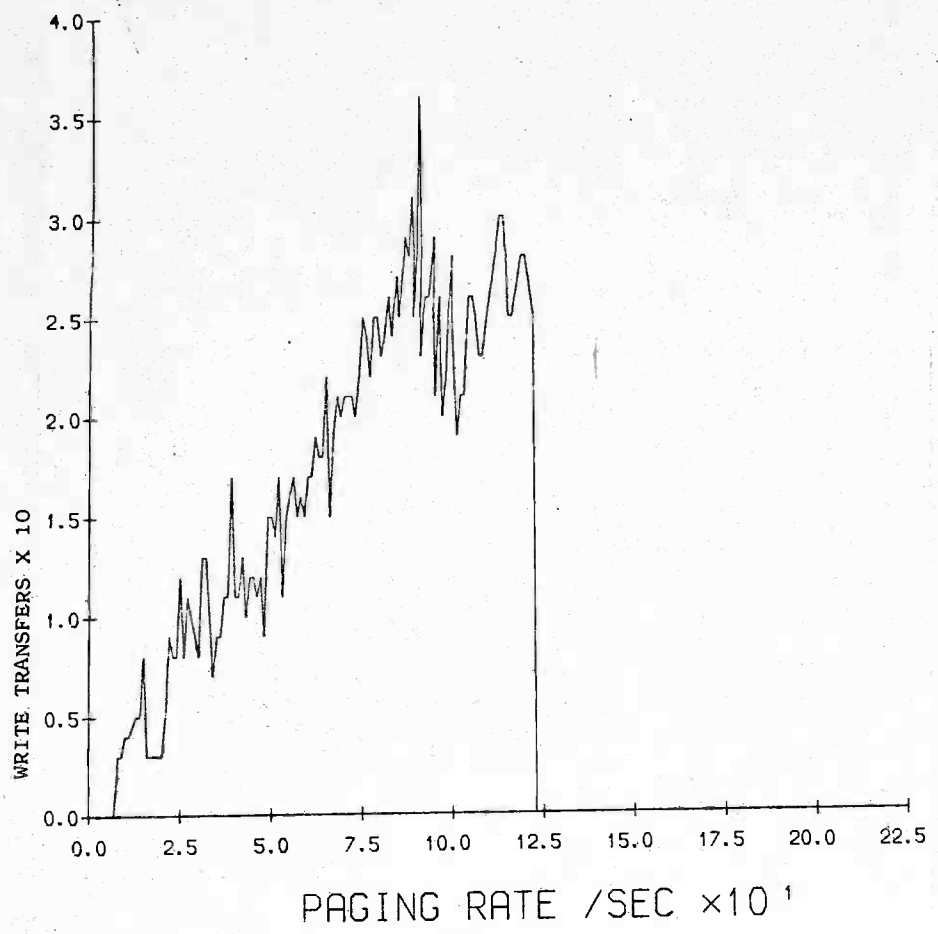
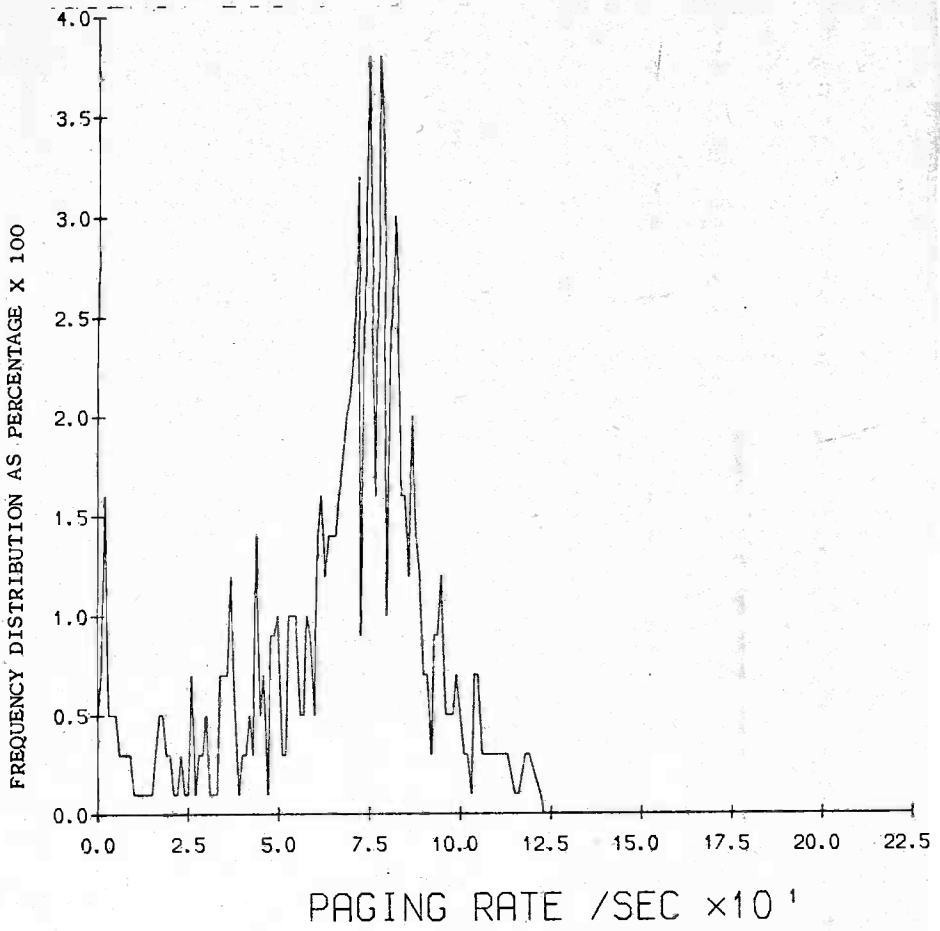


Figure 5.13 (g)

EXPERIMENT L



found in the larger main memory configurations. The effect of the channels is for the delay times to show steeper slopes in the single channel case i.e. when only one channel is in use the paging delay time is more sensitive to the paging rate existing at the time the request is issued. It also may be noted that the average delay for write transfers is very insensitive to the paging rate showing very little increase as the paging rate increases. Similarly, in the WSR runs the average wait time per preloaded page lengthens very little with the increase in paging rate. However, in all the runs the mean wait time for a page fault on the drum shows a very steady and much steeper increase as the paging rate goes up.

Considering the types of transfers which are making up the paging rate it may be seen that as the paging rate goes up, the number of page faults shows a distinct tendency to level off, however the number of write and preloading transfers continue to rise. This shows two things:

(1) The success of the main memory control algorithm in limiting the rate of page faults and hence completely eliminating any form of thrashing phenomena.

(2) Higher paging rates are due to process swapping rather than page faulting. This is further illustrated in the PDP algorithm runs when the number of preloading transfers are considered. In the case of the PDP algorithms this is equivalent to the number of process swap-ins. Also as the effective delay times for preloading and write transfers are not so very sensitive to the paging rate the times for process swapping under the WSR algorithm will not rise very much as the paging rate increases. This will be very critical when a substantial number of processes do very little work when they arrive in main memory (in terms of CPU use) but still have to pay the price of loading a large working set before any work may be performed.

Throughput

The throughput observed on EMAS during the experiment runs is shown in Table 5.27. The throughput is defined as the number of interactions completing in unit time. This is also proportional to the number of main memory residencies completing in unit time, the ratio being determined by the scheduling algorithm as seen earlier (Table 4.1). The factor most influencing the throughput is seen to be the algorithm

Table 5.27

Throughput Rates

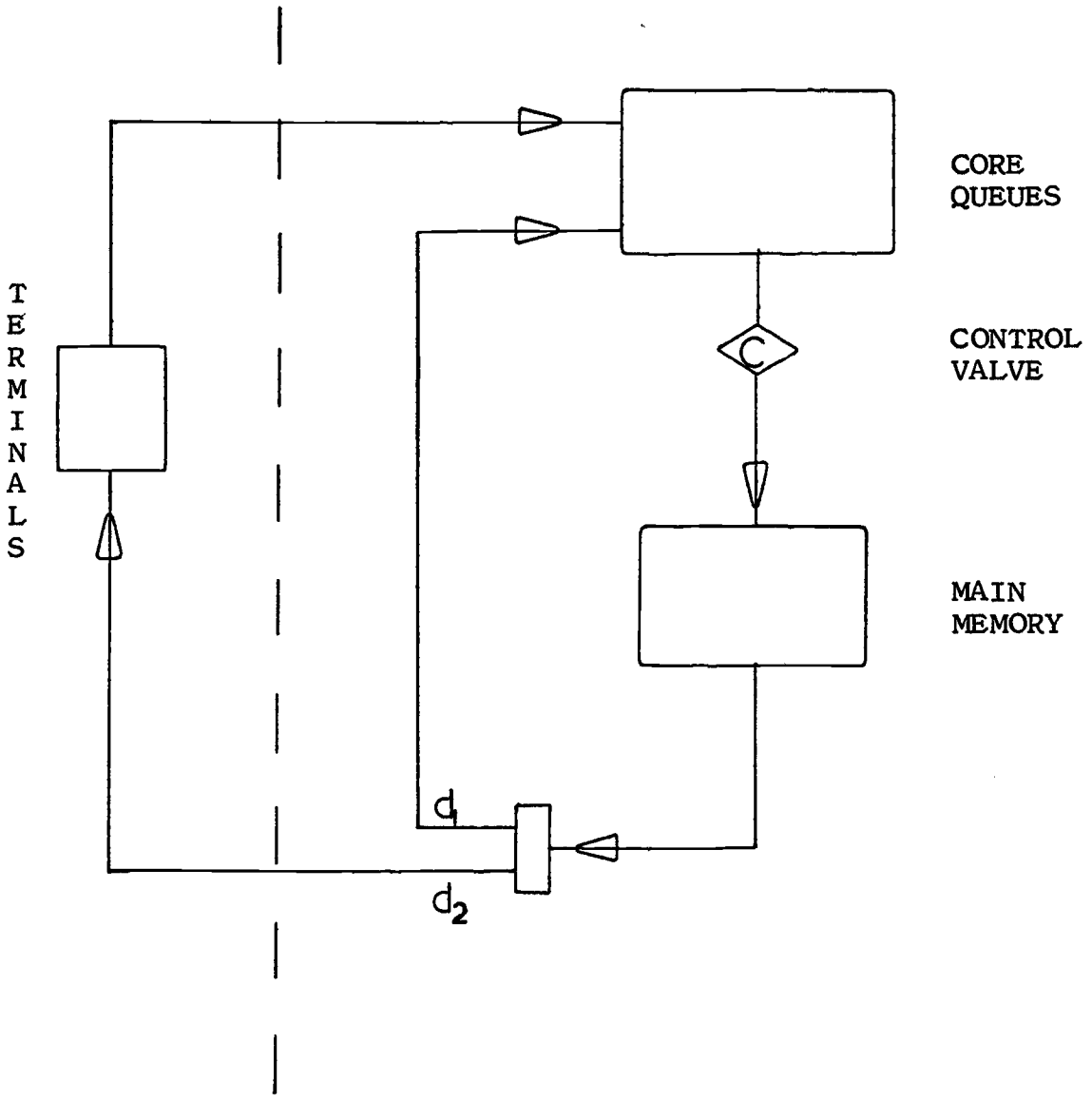
ESP RUN	INTERACTION COMPLETION RATE	RESIDENCY COMPLETION RATE	RATIO RESIDENCY/INTERACTION
A	1·60	2·36	1·5
C	1·44	2·17	1·5
E	1·50	2·25	1·5
G	1·25	1·94	1·6
I	1·36	2·03	1·5
K	1·21	1·80	1·5
B	1·23	2·60	2·1
D	1·12	2·42	2·2
F	1·10	2·33	2·1
H	0·97	2·12	2·2
J	1·02	2·15	2·1
L	0·91	1·91	2·1

used (Table 5.28) with the WSR algorithm giving a better throughput than the PDP algorithm by about 1/3 of an interaction a second. Similarly the greater the main memory size the greater the throughput. Both of these major factors - algorithm and main memory size - also had very significant effects upon the response times, but the effect of the number of channels upon the response times was much less than these two. However, in the case of a throughput measure, the effect of having two channels rather than one is much more significant.

Conclusion

The effect upon system performance of the three factors chosen for the experiment has been quantified in terms of three common performance measures, and the system phenomena contributing to these performance levels investigated. The manner in which the system functions can be summed up in terms of the simple model shown in Figure 5.14. When processes wake up, they enter the core queues and are held there until allowed into main memory by the control valve (C). The main characteristic used to discriminate between processes is their estimated main memory requirement. The rate at which the control (C) allows processes to enter main memory is determined by the rate at which

Figure 5.14


CORE
QUEUES
CONTROL
VALVE
MAIN
MEMORY
TERMINALS
C
d_1
d_2
ASLEEP
AWAKE

processes leave main memory (if it is assumed that work always exists in the core queues). This rate is therefore influenced by main memory size rather than the number of channels available, and least by the algorithm used (Table 5.29). The rate at which processes leave main memory will not be dependent upon the total number of processes in the core queues (though it will be affected by the ratio of the numbers in the different core queues and hence the balance of the workload between various classes of work). When the system reaches a state such that there is always at least one process on each core queue, then the throughput of the system may be considered to be totally independent of the number of processes active on it. The interference of processes with each other in such a state will be reflected in the core queue wait times which will dominate the response times.

When processes leave main memory they either go back to sleep, or re-enter the core queues to await a further quantum of time in main memory. If the balance in the workload is assumed fixed, then the ratio number of processes following each path (d_1 : d_2) is determined by the category table scheduling. If the category limits are never reached and each member of the multiprogramming set is

Table 5.29

ANOVA Table for the Average Main Memory Throughput - Residency Terminations per Second (Mean = 2·17)

SOURCE	AVERAGE EFFECT	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARE	MEAN SQUARE RATIO
MEMORY					
7/8 3/4	-0·23)				
3/4 5/8	-0·19)	0·115	2	0·58	77·6 ****
7/8 5/8	-0·41)				
CHANNELS	-0·227	0·51	1	0·51	69·2 ****
ALGORITHM	0·163	0·027	1	0·027	36·0 ****
HIGH ORDER FACTORS (ERROR ESTIMATE)		·004	7	0·0003	
TOTAL		0·20	11		

**** SIGNIFICANT AT THE 99·9% level (F - test)

allowed to run to completion ($d_1 = 0$), then the system essentially runs in batch mode. The more the category limits are enforced (i.e. d_1 increases) then the greater the level of time-sharing of the main memory resource. However, each process swap has a large paging overhead associated with it (WSR - 593 milliseconds, PDP - 248 milliseconds + page faulting time) so though time-sharing would be expected to lower response time, when it has passed a certain point it will have a detrimental effect on the response, as seen in the comparison between the WSR and PDP algorithms. Other ways of improving the response time are to reduce the time spent in page wait (either by adding channels or possibly by preloading) and increasing the level of overlap of processes by increasing the multiprogramming level by increasing main memory size. There are two main differences between the two algorithms employed in the experiment:

(1) The placement of processes in categories which will influence the ratio $d_1 : d_2$ and also the mean time a process is allowed to spend in the MPS.

(2) The paging delay times and hence again the time any process spends in the MPS.

These differences between the algorithms and certain other factors which might influence the system performance are further investigated in the next chapter by means of a simulation model.

Chapter 6

In this chapter some of the factors which influence the system's performance are investigated further by means of a simple simulation model. The results from the EMAS performance experiment are used to calibrate and validate this model, thus gaining some confidence in the predictions obtained from it.

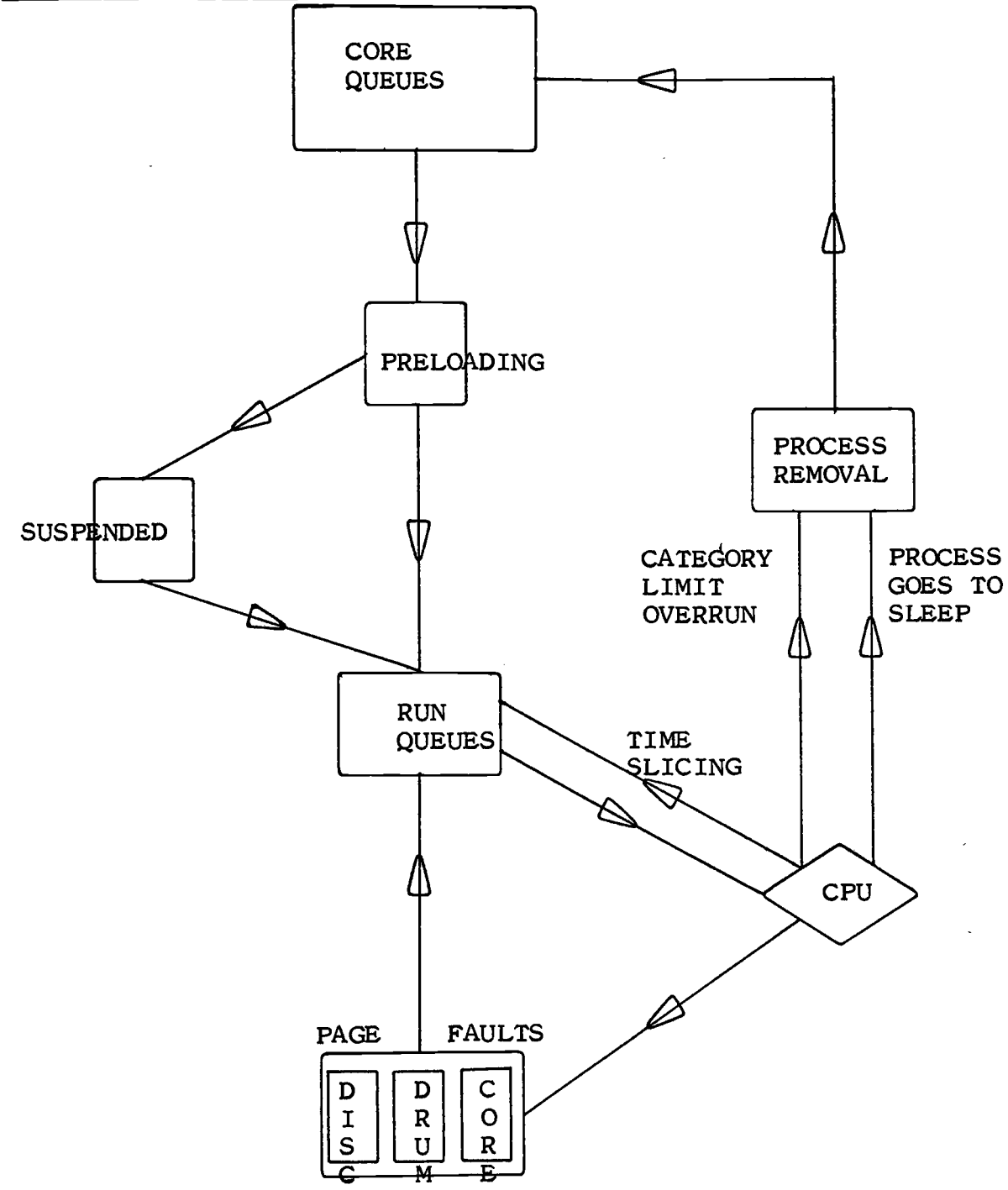
Structure of Model

The model used is a simple discrete event simulation [Leroudier and Parent 1976] written in IMP [Stephens 1974]. As is the case with all models it is an abstraction and a simplification of the real system. In this case only the main-secondary memory subsystem is modelled in any detail (Figure 6.1), this subsystem having been identified in the previous chapter as being crucial to the performance of the system as a whole.

The choice of using simulation rather than a variation on the queuing network models often used in evaluations of this type of system was based on a desire to reflect accurately the working of the subsystem modelled. This implies that the model would

Figure 6.1

SUBSYSTEM MODELLED



have to include a representation of:

Supervisor overhead.

Variability in user process behaviour.

The effects of bulk and single transfer requests to the secondary memory.

The existence of blocking phenomena .

Though some of these phenomena have been included in various mathematical queuing theory models [Baskett et al. 1975, Baudet et al. 1975, Potier 1977] no technique yet exists which allows for the inclusion of all of them in a single model. However, every attempt has been made to keep the simulation itself simple and hence tractable. It is also hoped that the model will be extendable and be of use in the evaluation of aspects of the system's behaviour not covered here.

The model consists of three logical units:

1) Simulation support.

2) Simulation of resident supervisor algorithms.

3) Prediction of hardware device and user process characteristics.

1) Simulation Support

This component takes care of all the facilities, associated with the simultaneity of process activity and timing of events, normally provided by special purpose simulation languages or packages [Dahl and Nygaard 1966, Dimsdale and Markowitz 1964]. All the synchronisation and timing of events in the simulation is controlled by a central time queue which holds all events which are known to be due, ordered in ascending time of occurrence. Primitives are provided by this component to take care of the placing of events on the queue and removing events from the top of it when they are about to 'happen'. This component also manipulates the two simulation timers. One of these is the simulation clock holding the current value of simulated time, and the other is the simulation alarm clock holding the time at which the first event in the time queue is due to occur. The simulation clock used throughout had a precision of one millisecond, though no assumptions are made in any part of the

simulation about any specific granularity of time. Monitoring of significant simulation variables and the outputting of results is also handled in this module.

Simulation of Resident Supervisor Algorithms

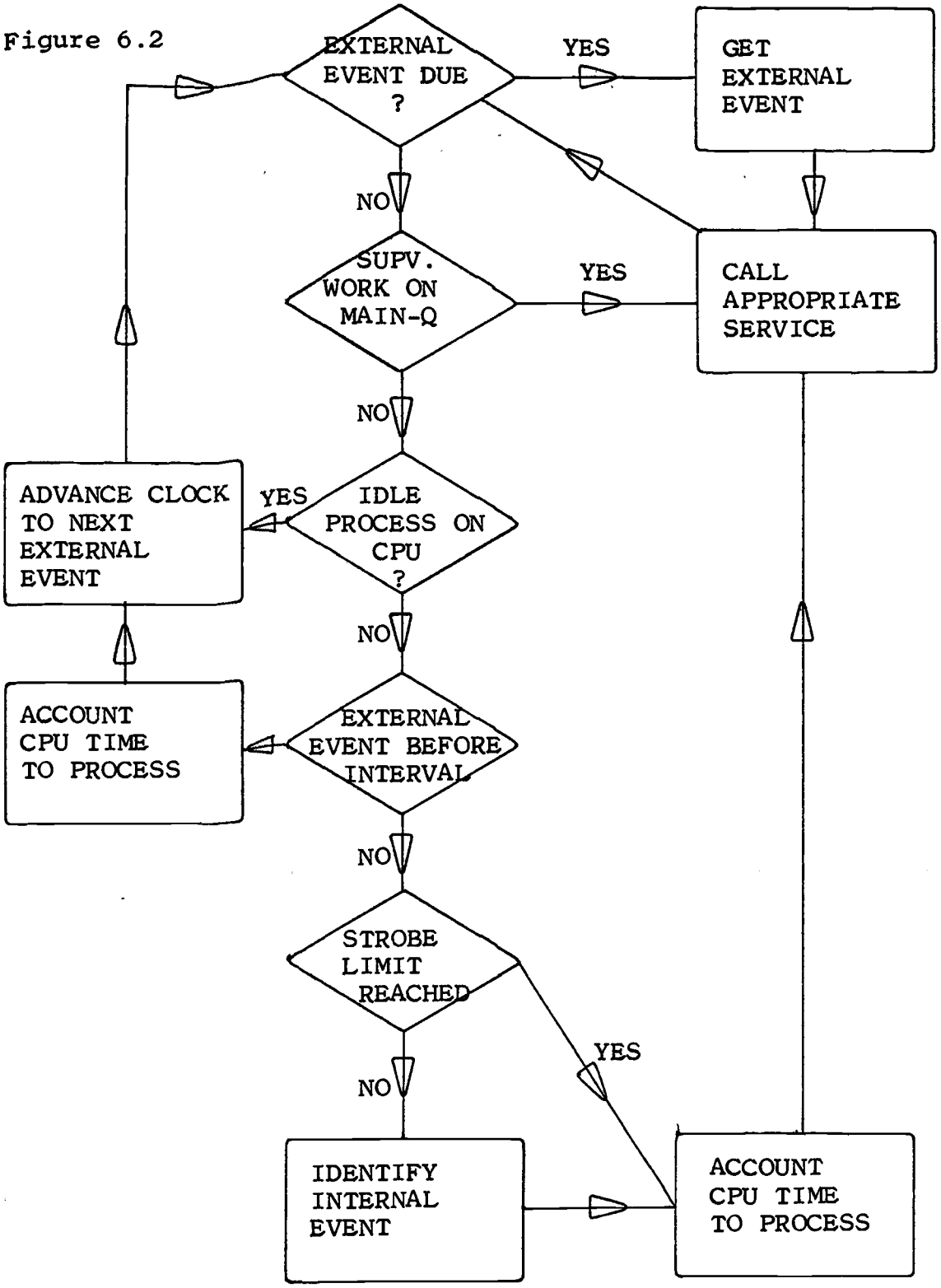
This component mimics certain of the activities of the EMAS supervisor and consists of a kernel and a set of supervisor services. The services control the operation of simulated hardware devices and the allocation of a set of simulated resources.

Supervisor Kernel

As in the real system the kernel takes care of dispatching supervisor services and fielding 'external' interrupts by translating these into requests on services. All communication between services takes place via a central parameter passing area and all supervisor services which have outstanding requests awaiting them also have an entry in the kernel's MAIN-Queue.

The kernel itself consists of an endless loop (Figure 6.2) which continually checks whether an interupt is currently pending by comparing the values

Figure 6.2



SIMULATED KERNEL - FLOW DIAGRAM

of the two simulation timers. Any outstanding interrupts found are translated into requests upon appropriate services. This check for interrupts takes place after each service call, thus mimicking the behaviour of the real system in which supervisor services run uninterruptably with interrupts only being taken between calls on services. When there are no outstanding external events the kernel main queue is inspected to see if there are any current requests for supervisor activity. If so, the first service in the queue is called, and a check for interrupts again made. When all supervisor work is complete for the present, then a check is made to see if the current CPU process is the idle process. If it is, then the simulation clock is advanced to the time of the next event, and the next external interrupt is serviced. If a normal user process holds the CPU then a check is made to see it is not overrunning any of its category CPU limits. If so, a request is placed for an appropriate service. If not, then a check is made to find out if the CPU-process can advance the simulation clock as far as its next process defined event (pagefault, SVC, sleep or end of time slice - 'internal interrupts'), if this is possible then the clock is moved to this point and an appropriate supervisor request issued. If an external interrupt is due before the next user process generated event

then the external interrupt will take precedence, the user process advances the simulation clock as far as the external event time, is credited with using that CPU time, and the external interrupt is serviced.

Supervisor Services

All process scheduling on the simulated system is based upon a category table similar in format to that used in the real system but making no reference to secondary memory allowances (Table 6.1) as secondary memory capacities are not included in this simulation. Each process has an associated entry in the process list. This entry holds such items as the process' current category, working set size, main memory allocation, CPU time obtained and process status. The processes are moved between scheduler queues by the supervisor services mimicking the algorithms used in the real system (Figure 6.3) and handling such functions as:

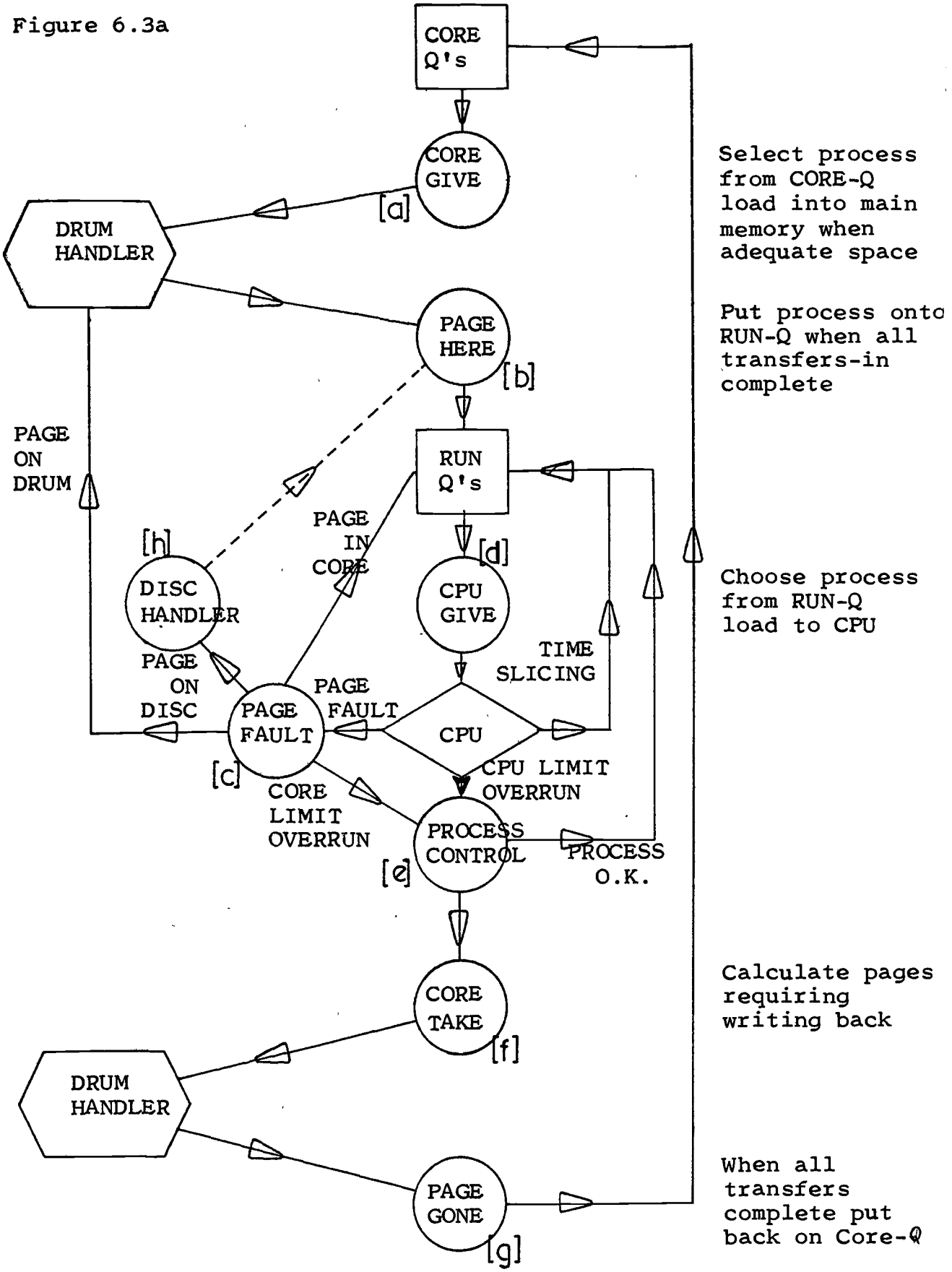
a) Entry to the MPS - selection of processes from the core queues as memory becomes available and organising requests for process loading transfers.

Table 6.1

SIMULATION CATEGORY TABLE VALUES

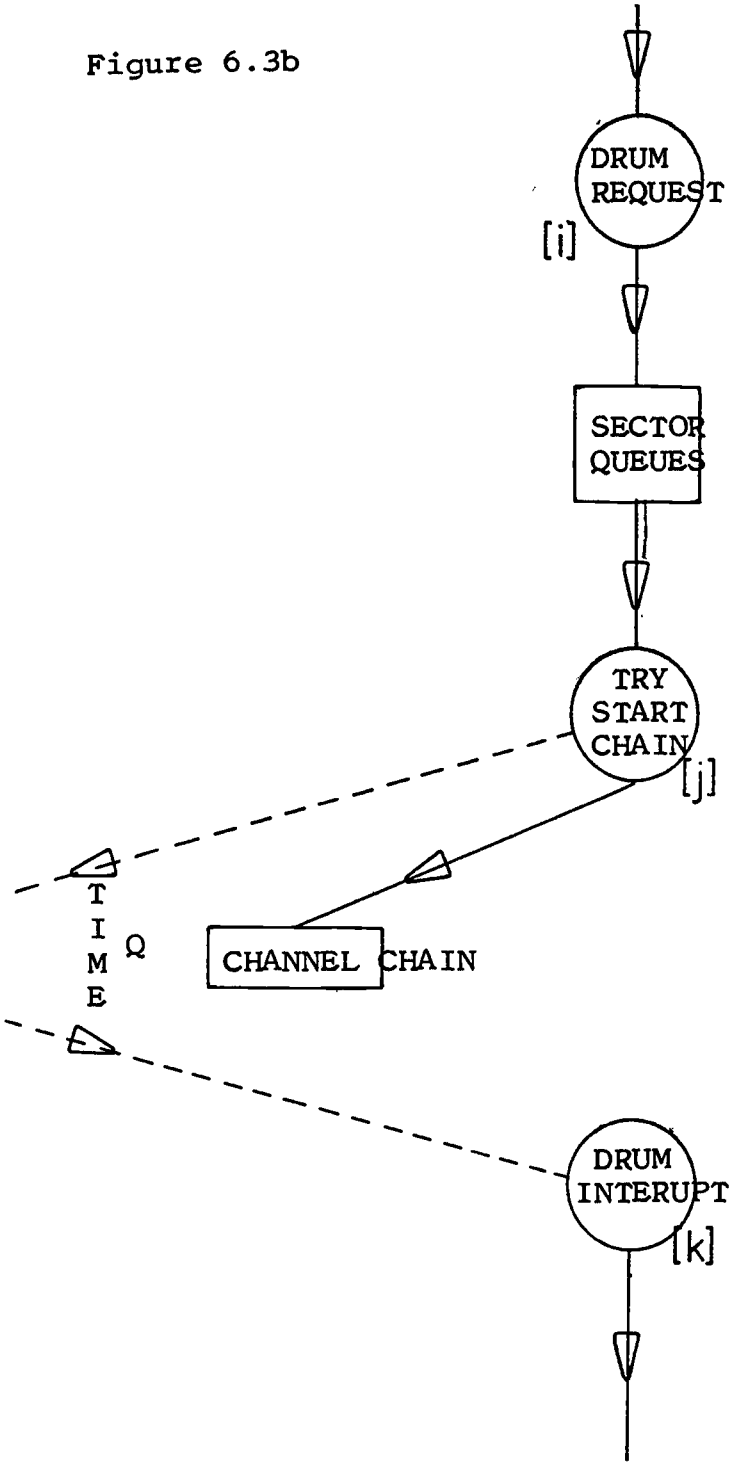
CAT	PRIORITY	MAIN MEMORY ALLOWANCE (PAGES)	RESIDENCY CPU TIME (SECONDS)	STROBE INTERVAL (SECONDS)
1	1	50	1·0	0·125
2	1	20	0·5	0·5
3	1	30	1·0	1·0
4	1	50	2·0	0·5
5	1	20	0·5	0·5
6	4	20	4·0	1·0
7	4	20	10·0	1·0
8	1	30	1·0	0·5
9	4	30	10·0	1·0
10	4	30	6·0	1·0
11	2	40	1·0	1·0
12	4	40	10·0	1·0
13	5	40	12·0	1·0
14	2	50	1·0	1·0
15	4	50	10·0	1·0
16	5	50	10·0	1·0
17	3	60	2·0	0·5
18	4	60	7·0	0·5
19	5	60	5·0	1·0
20	3	62	2·0	0·25

Figure 6.3a



SIMULATION MODEL - SUPERVISOR SIMULATION

Figure 6.3b



Put in sector-queue according to priority demand > preload > write

Construct channel chain. Place completion interupts in Time-Q

Identify processes having transfers in chain and notify them

SIMULATION MODEL - DRUM HANDLER

b) Fielding replies to page-in requests and placing the process in its run queue when it has completed all of its transfers.

c) Handling page faults and organising any transfer requests which may arise.

d) Allocation of the CPU - choosing a user process from the run queues or selecting the idle process if no such user process is available.

e) Enforcing the category table resource limits and selecting a new category when a process is being removed from main memory.

f) Removal of a process' pages from main memory during strobing or process removal from main memory.

g) Fielding replies from page-out requests and placing the process back into the scheduler queues when all such requests have completed.

The only hardware devices currently simulated are the drum (secondary memory) and disc (tertiary memory). The disc is not itself modelled in

any detail but is only represented by a delay with its handler (h) merely placing a page-in reply in the simulation time queue for the page here service (b). However, the drum, which has been found to carry considerably more traffic, and is held to be much more critical, is modelled in greater detail with three separate services which take care of:

i) The handling of drum transfer requests - putting them into sector queues according to an appropriate priority scheme.

j) The starting of channel chains, composed of requests removed from the sector queues, when the channel is found to be free, and placing interupts in the time queue signifying the termination of demand page reads and channel chains.

k) The fielding of interupts and firing off replies to the page here (b) and page gone (g) services as necessary.

Each time a supervisor service is called it advances the simulation clock by an amount corresponding to the overhead imposed by that service. The overhead times consist of a constant, plus in some

cases an amount which depends upon the number of pages being processed (for drum requests, drum interrupts, process entry to and exit from main memory). In addition to this, at the end of each burst of supervisor activity (i.e. just before mounting a user process or idle process to the CPU), a further supervisor overhead proportional to the preceding supervisor burst is added to represent supervisor time spent servicing items such as communications, secondary memory capacity allocation, supervisor calls etc. which are not explicitly included in this simulation. Each of the services has embodied in it, calls on an event trace monitor mirroring the calls made in the real system. It would be possible to use such data to obtain performance measures on the simulation, however this has only been used to check the correct working of the supervisor algorithms implemented in the simulation. In that area this feature has proved invaluable.

Process/Device Behaviour Definition

The third logical unit in the simulation consists of a set of functions which are called from various points in the supervisor simulation and define the characteristics of hardware devices and user processes existing in the simulation.

Hardware Devices

The behaviour of hardware devices consists of a set of functions which handle:

1) The prediction of the completion time of a demand read from disc. The result is drawn from a random number function with an appropriate distribution, the mean of which is currently taken as 210 milliseconds (the average of all disc page faults over all the experimental runs).

2) The sector corresponding to any particular drum request. The result here is drawn from a random number function evenly distributed in the interval [1, maximum number of sectors] so there is an equal chance of each request going to any sector.

3) The latency time before the first drum transfer in any drum channel chain. This is drawn from a random number function evenly distributed over the interval [0, maximum latency time].

4) The transfer time of any drum page.
This is currently a constant.

The above totally define the characteristics of the hardware devices simulated.

Process Characteristics

The behaviour of a process is defined by a set of functions which predict the type and timing of internal events and the behaviour of process working sets. These functions take care of:

1) Predicting the next significant internal event which will occur for this process. Currently, internal (process defined) interupts may be either pagefaults or sleeps. The event is chosen from a table containing the event types in the correct proportions with a separate table being held for each category.

2) Whenever a pagefault occurs, the type of fault (disc, drum or in main memory) is determined from a table, holding the three fault types in appropriate proportions.

3) The user CPU time which will pass before the next internal event. This is determined from a lifetime function for each category. The lifetime function relates the expected CPU time until next event to the number of pages currently used by the process (i.e. any wasted preloaded pages are not taken into account). The functions used (Figure 6.4) are extracted from the event trace data and reflect the average effect of all the processes running in each category. This differs from the original lifetime function [Belady and Keuhner 1969] (used in many mathematical models of this class of system) which relates the mean time between pagefaults to the number of pages owned by a programme. A system observes and reacts to the behaviour of a process which is a collection of co-operating programmes. The approach taken here though necessarily crude is, however, more realistic than using a simple (monotonic) lifetime function. The variance within the lifetime functions is represented by means of random number function which defines the variance to be added to or subtracted from the lifetime function value as a fraction of the

Figure 6.4a

LIFETIME FUNCTIONS

(WORKLOAD 1)

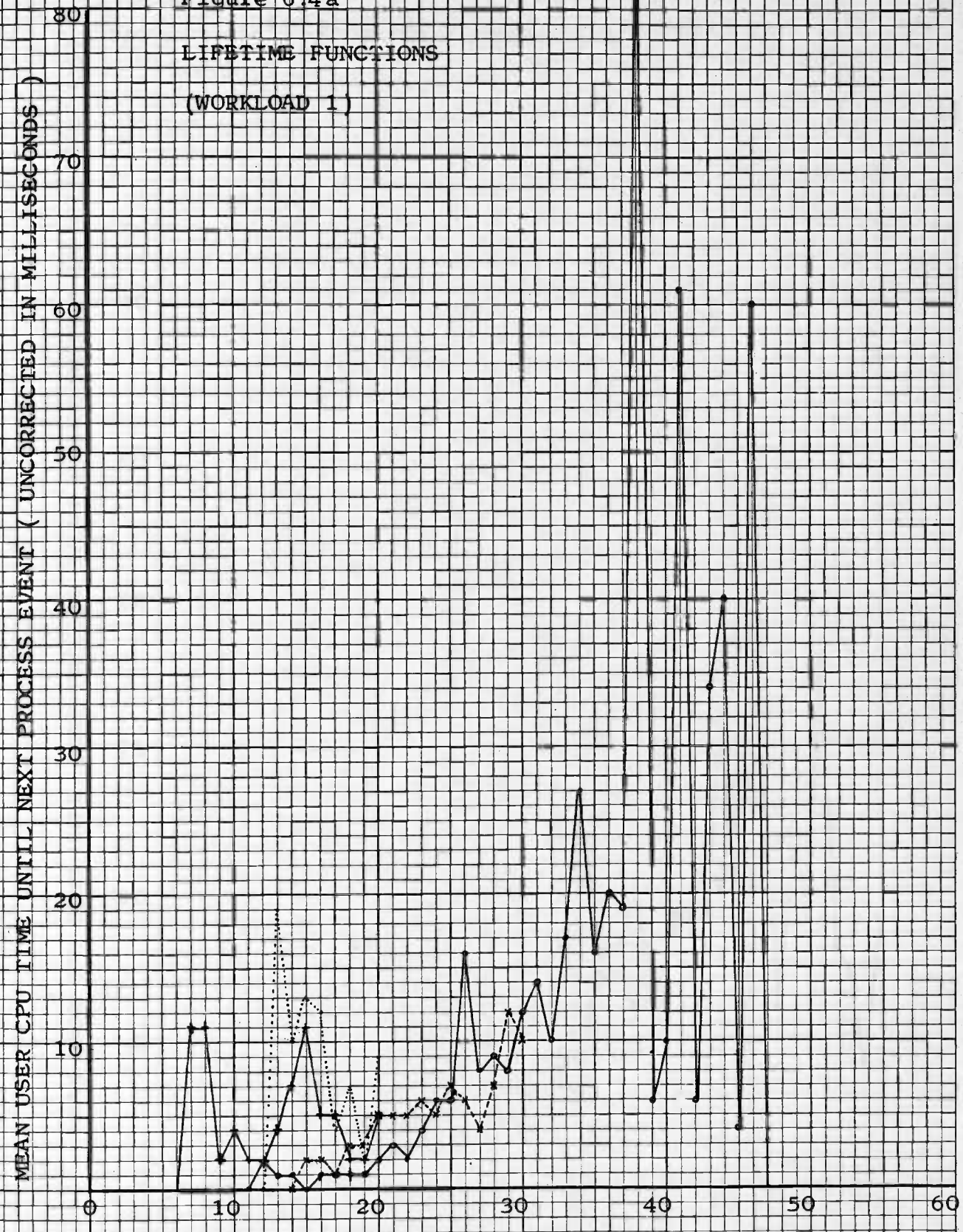
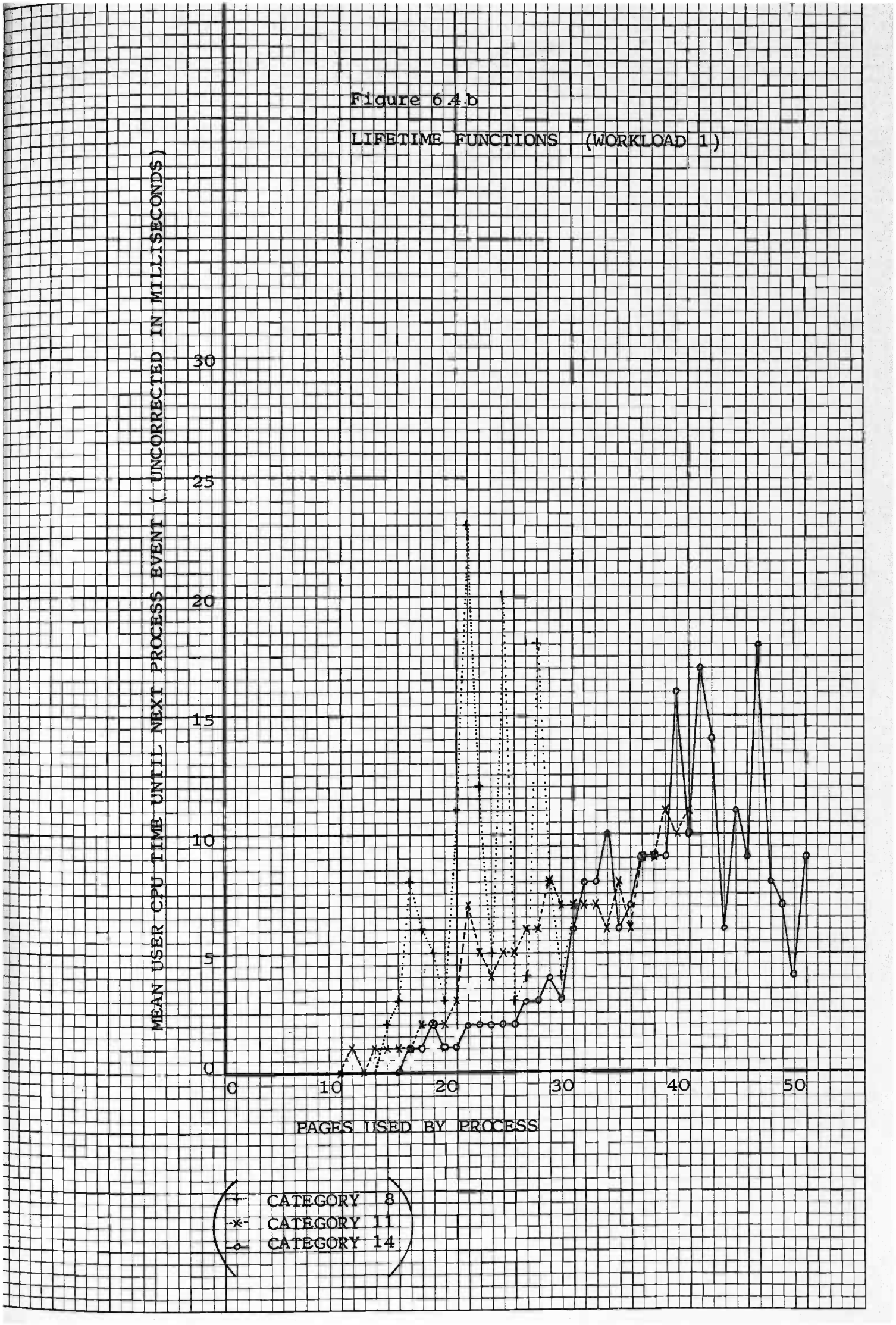


Figure 6.4.b

LIFETIME FUNCTIONS (WORKLOAD 1)



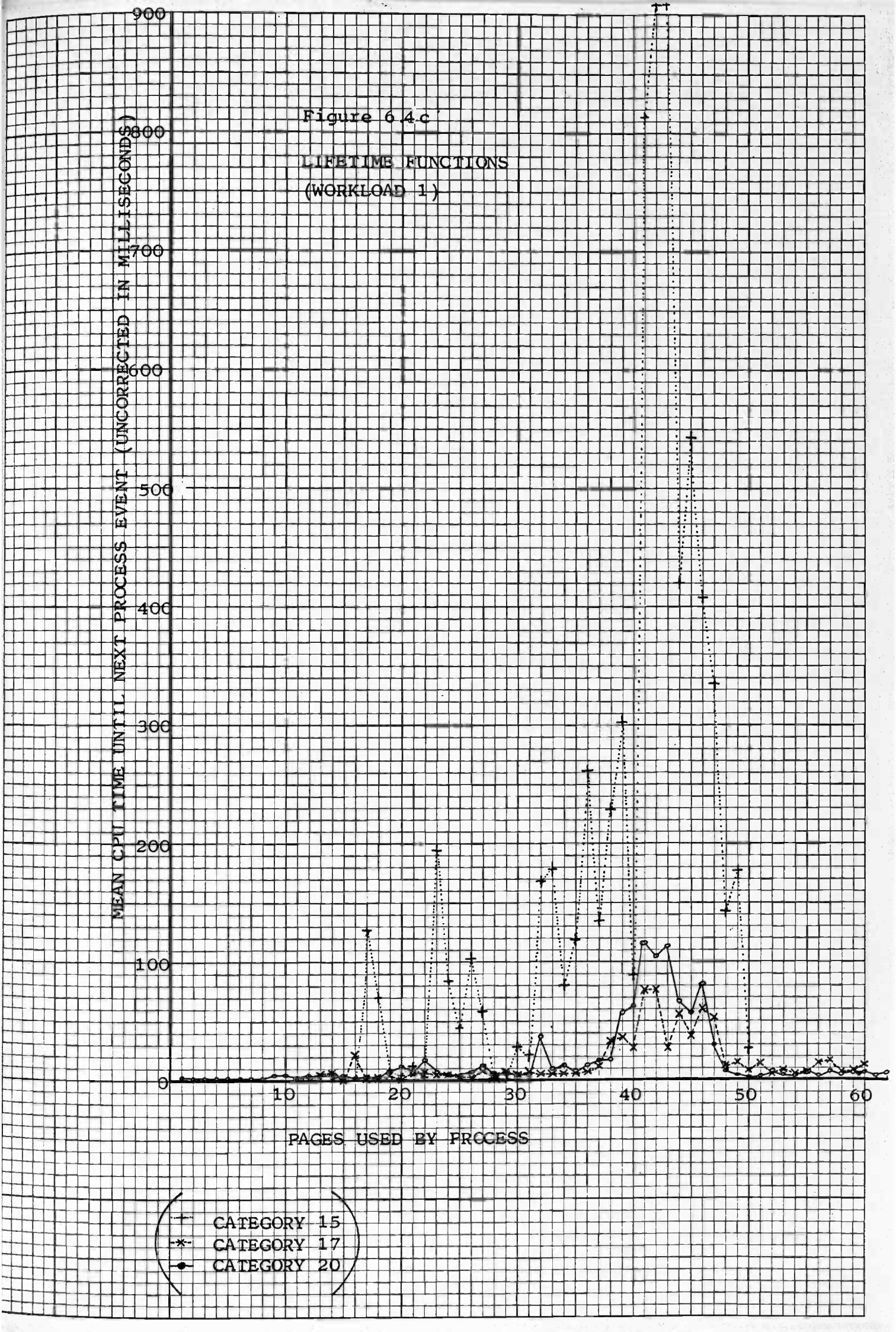


Figure 6.4c
LIFETIME FUNCTIONS
(WORKLOAD 1)
MEAN CPU TIME UNTIL NEXT PROCESS EVENT (UNCORRECTED IN MILLISECONDS)
900
800
700
600
500
400
300
200
100
0
10
20
30
40
50
60
PAGES USED BY PROCESS
CATEGORY 15
CATEGORY 17
CATEGORY 20

Figure 6.4d

LIFETIME FUNCTIONS (WORKLOAD 2)

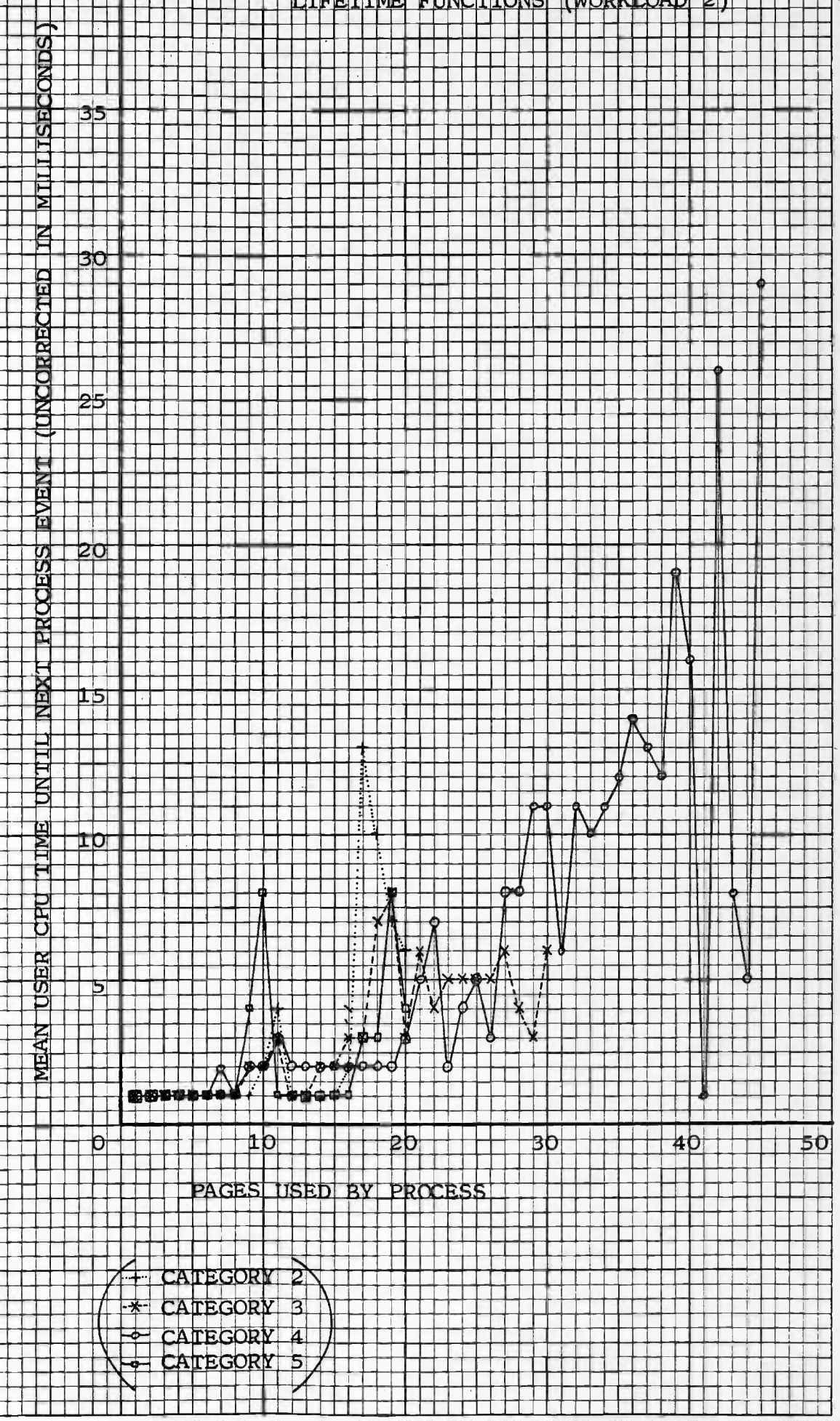


Figure 6.4e

LIFETIME FUNCTIONS (WORKLOAD 2)

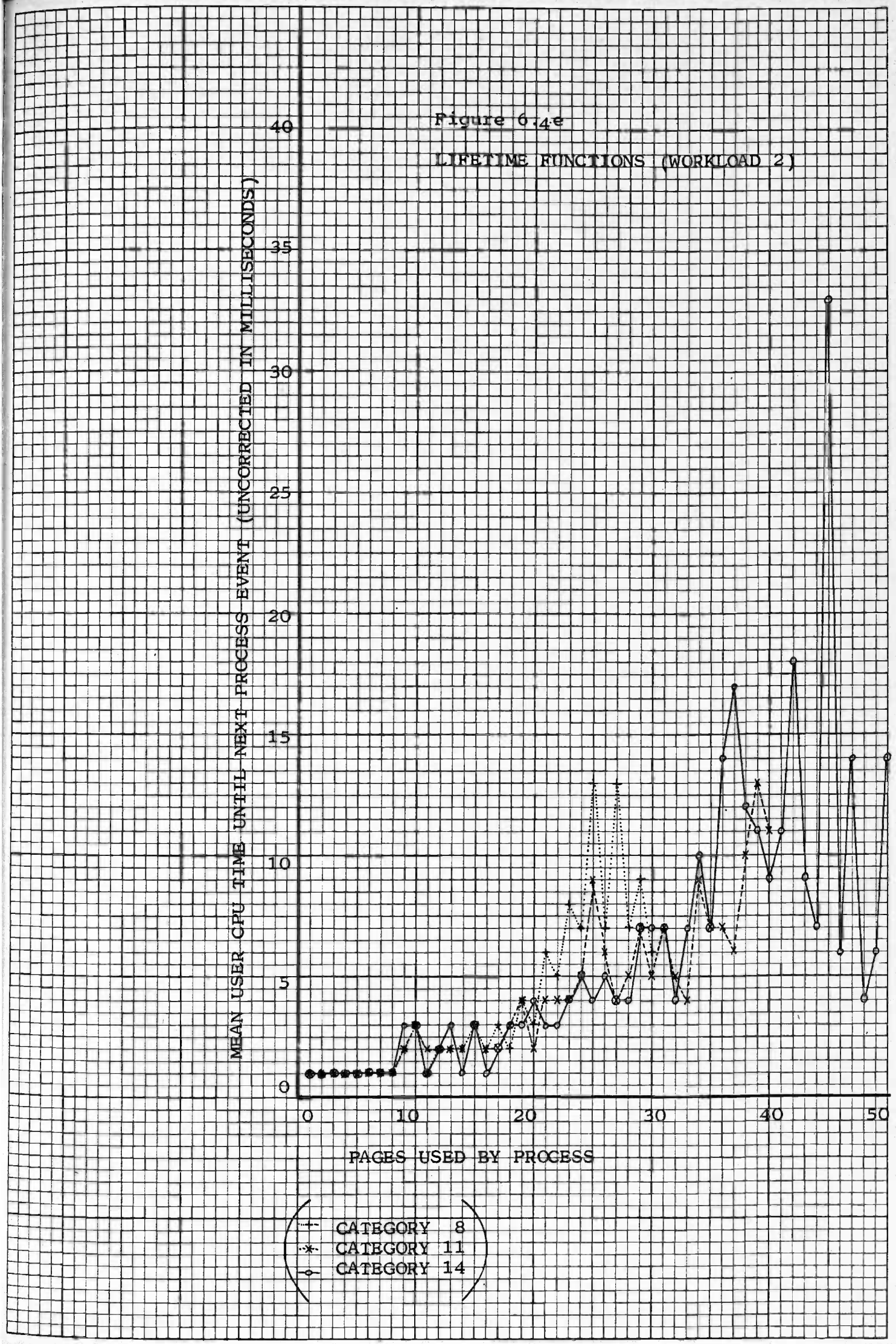
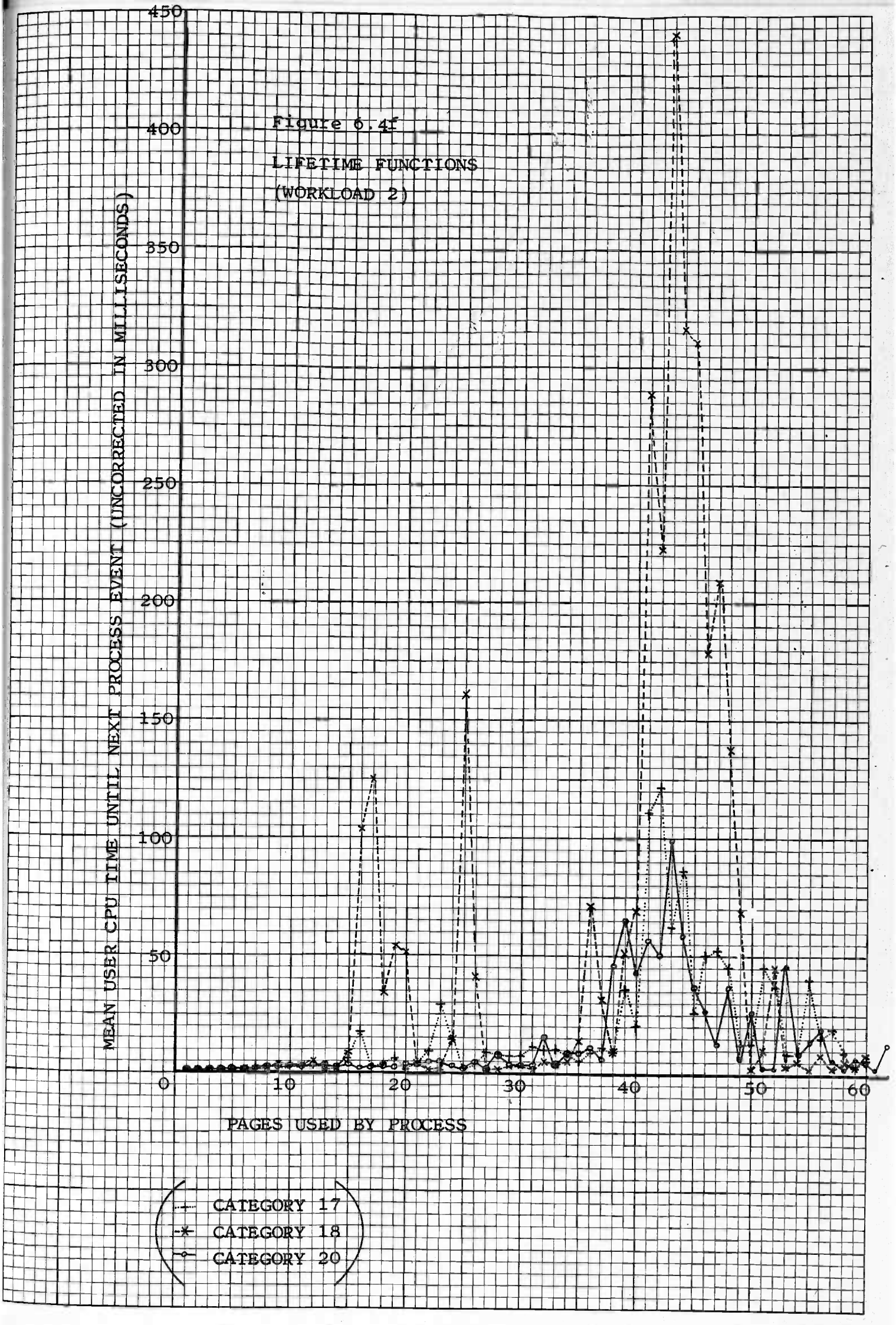


Figure 6.4f

LIFETIME FUNCTIONS
(WORKLOAD 2)



current value i.e.

CPU to event = f(CAT, PAGES) + x* f(CAT, PAGES)
where x is the random number function (mean = 0)
representing variance in the CPU times.

The other functions in this component represent movements within a process working set. These define:

4) The core set size when the process enters main memory. This is defined as the number of pages which will be prepaged and used. This will always be equal to one when WSR is not being used. The result is chosen from a random number function with an appropriate mean (different for each category).

5) The number of pages which are preloaded and not subsequently used. This is calculated from a random number function which represents the number of wasted preloaded pages as a fraction of the number of usefully preloaded pages.

6) The number of pages discarded at each strobe carried out on a process. For the first strobe in a main memory residency this will always be the number of wasted preloaded pages (if WSR is being used) or zero (if PDP is being used). For subsequent strobings during the residency this number is a simple function of the number of pages held by the process i.e.

Pages discarded during strobe i of residency =

```
( 0                 if i=1 and PDP employed
( Wasted Preloads   if i=1 and WSR employed
( PAGES // 16       if i>1
```

7) The number of pages which are preloaded but require no transfers (i.e. shared pages). This is obtained from a random number function giving the fraction of pages about to be preloaded for this process which will be shared.

8) The number of pages written to during any residency and now requiring writing back to the secondary memory. This is determined using a random number function representing the fraction of useful pages held by the process which will require

writing out.

Pages requiring writing out =

(x) * (number of useful pages held by process).

Using the Model

In this investigation the workloads used in the simulation runs consisted of a set of 50 processes which were each permanently assigned to certain categories (Table 6.2). That each process remained in its chosen category throughout the simulation was achieved by making all the category transitions held in the category table point to the process current category. This maintains a fixed balance within the available workload, in an attempt to make the balance of categories passing through main memory reflect these proportions only one core queue was used. This avoids the priority scheme which normally operates when choosing which process should be next to enter the multiprogramming set. During the simulation the system is studied in a saturated state i.e. each of the core queues which would have been used with the priority scheme to always have at least one process on them when inspected. This means the balance of category priorities passing through main memory would always be in the fixed ratio of the relative rates at which the

Table 6.2

PROCESS ASSIGNMENTS TO CATEGORIES IN SIMULATION

CATEGORY	PROCESSES IN THIS CATEGORY	
	WORKLOAD 1	WORKLOAD 2
1	0	0
2	1	0
3	2	2
4	1	1
5	2	8
6	0	0
7	0	0
8	22	12
9	0	0
10	0	0
11	11	10
12	0	0
13	0	0
14	6	10
15	1	0
16	0	0
17	3	4
18	0	1
19	0	0
20	1	2
TOTAL	50	50

priority queues were serviced. This does not reflect the real system in which the balance of the load will vary over time causing some queues to be empty when due to be serviced and thus disturbing the priority balance. The single core queue solution was considered to be the simplest and best way of attempting to keep the balance of categories fixed - a modified priority scheme was attempted but did not prove as successful as the single queue version. The balance of categories used in the workloads was based on observations made in the experiment runs (Tables 4.3, 4.4, 4.5) slightly modified during the calibration process. Two workloads are used:

Workload 1 uses the lifetime functions obtained from group 1 experiment runs.

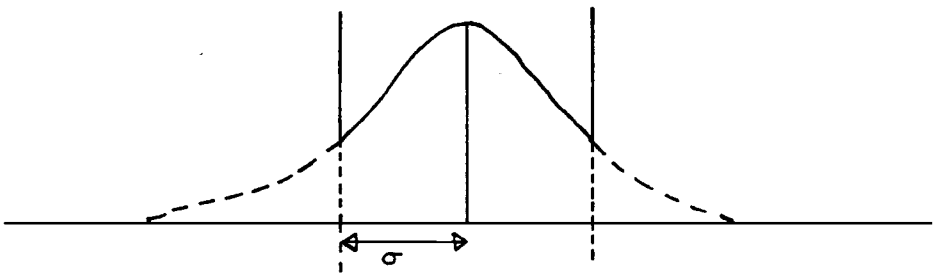
Workload 2 uses the lifetime functions obtained from group 2 experiment runs.

As part of the policy of keeping the simulation simple, no attempt was made to mirror exactly the often complex distributions found on the system. All the distributions used in the random number functions (except where a uniform distribution was being used) were modified normal distributions. This modification consists of removing the tails of

the distribution beyond one standard deviation from the mean, and piling up this part of the distribution at the two cut-off points (Figure 6.5).

In the area of secondary storage all requests to the drum were ordered in the sector queues according to a priority scheme which gave demand page reads priority over prepage reads, which in turn had priority over writes. However no attempt was made to model secondary storage capacity (i.e. contention for drum space) or the fact that the secondary memory consisted of several drums with a priority scheme between them. The secondary memory may be thought of as one large drum. This causes writes to take slightly longer and the other two forms of transfer to pass through slightly quicker. It would have been possible to have modelled the priority scheme differentiating between physical drums using a probability function which would have associated a probability for each request with each of the separate devices. However, in an attempt to keep the model simple, and because of the lack of data in this area, this was not done. It would also have been possible to have modelled twin channel operation by defining a probability function which would have decided when overlapping double channel transfers were possible, but similarly this was not done. The only transfers involving the disc storage

Figure 6.5



MODIFIED NORMAL DISTRIBUTION ADOPTED

IN THE SIMULATION

are for demand page reads, no disc writes are simulated.

No virtual memory addresses are simulated in any way. A process working set is represented merely by the number of pages in it. The time until the next addition to working set being defined by the lifetime function.

Performance Metrics

The performance metrics chosen for this study are:

1) Mean CPU utilisations.

2) Mean Drum transfer rates.

No response times were applicable as no user sleep state is modelled. The model was calibrated for the two workloads using the data from experiment runs K and L. The workload was adjusted to make the mean CPU time per residency, mean number of pages per residency and percentage of pages preloaded, as well as the main performance metrics, as near as possible to the observed values. The supervisor overheads used for the various services was taken from an

average over those two runs rather than taking different CPU overheads for each workload. So, the only differences between the two workloads were those factors which defined user process behaviour.

In running the model a period equivalent to 100 residencies was allowed to remove start up effects. The model was then run for a period of 1000 process residencies and the performance metrics taken from an average over that period. This long period was necessary because of the highly non-homogeneous behaviour of user processes. Tests running the model over longer periods have shown that it has reached stability by this time i.e. no change in the observed metrics resulted from longer runs.

The confidence intervals in the simulation were obtained by a method suggested by Conway [Leroudier and Parent 1976, Badel and Zonzon 1976] in which the simulation run is divided into equal size blocks and the mean of each metric calculated over these blocks. These means are then used to estimate the variance of each metric, measured over the whole simulation run.

Effect of Main Memory Size

The model was used to investigate the effect of main memory available to user processes upon the chosen performance metrics. All other components which might affect the performance - workload, scheduling algorithms, secondary memory - remained fixed. The validity of the model may also be judged by comparing the observed values with those predicted by it.

The predicted values of the CPU time obtained by user processes (Figure 6.6a) always lie within 6·5% of the values seen during the performance experiment, though the model does tend to underestimate this metric in the higher memory size. This underestimation may be due to the mix of categories in the workload and the fact that the model contains no representation of time spent in the kernel state (this time is accredited to user processes in the empirical measurements). Workload 1 (using Working Set Replacement) is found to reach saturation at around 240 user pages with very little seeming to be gained (in terms of this metric) by increasing the memory size beyond this whilst keeping all other elements in the system constant. Workload 2 (using Pure Demand Paging) does not reach its saturation point so quickly and,

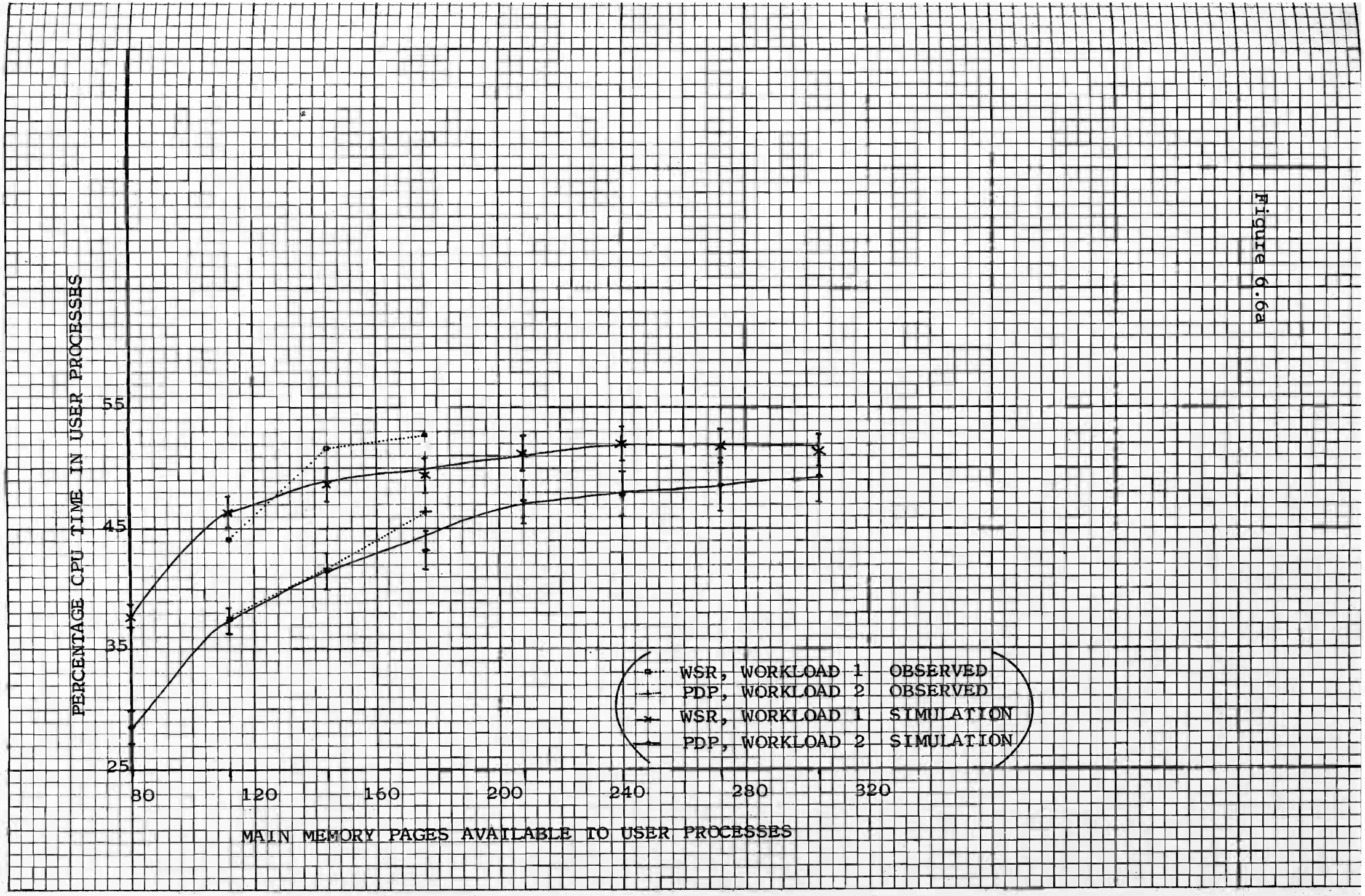

PERCENTAGE CPU TIME IN USER PROCESSES
55
45
35
25
80
120
160
200
240
280
320
MAIN MEMORY PAGES AVAILABLE TO USER PROCESSES
WSR, WORKLOAD 1 OBSERVED
PDP, WORKLOAD 2 OBSERVED
WSR, WORKLOAD 1 SIMULATION
PDP, WORKLOAD 2 SIMULATION

Figure 6.6a

though the gap between the two workloads diminishes as the main memory size increases, workload 2 always seems to give less time in user state than workload 1.

The values produced by the model for the time in supervisor state (Figure 6.6b) always lie within 10% of the observed values. The workload 1 results being a constant underestimation and not as good a fit as those obtained for workload 2 (the same is true for the user CPU time). This may be partially due to the fact that the overhead times associated with each supervisor function were averaged across the two workloads and that this procedure has favoured workload 2. In both cases the supervisor CPU time shows a steady monotonic increase as the memory size goes up.

The throughput on the secondary memory as predicted by the model always lies within 12% of the observed values for workload 1 (a consistent overestimation) but is within 4% for workload 2. As the main memory size goes up, the difference between the workloads increases, with workload 1 (using WSR) constantly achieving a higher throughput than workload 2. (Figure 6.6c).

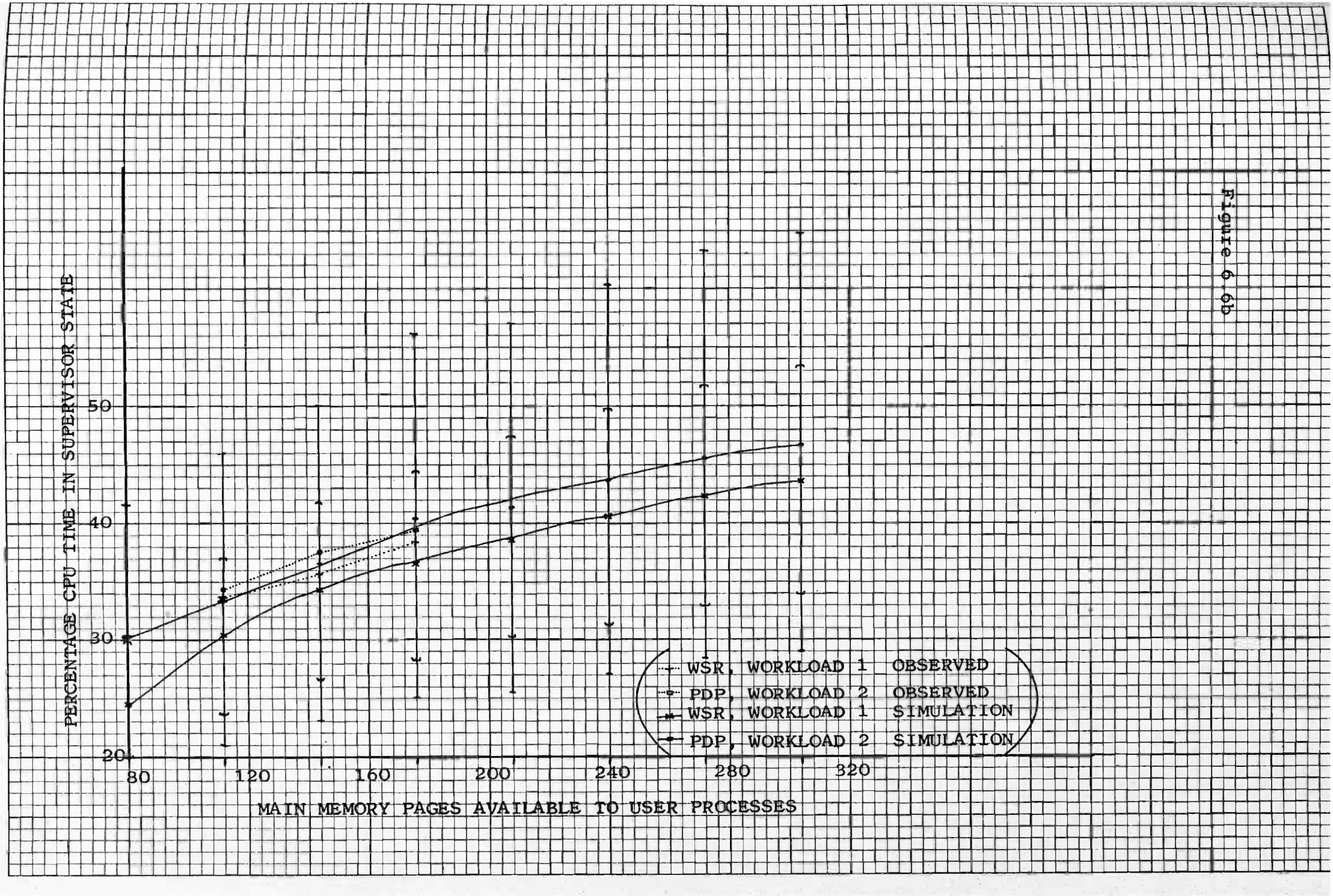

PERCENTAGE CPU TIME IN SUPERVISOR STATE
50
40
30
20
80
120
160
200
240
280
320
MAIN MEMORY PAGES AVAILABLE TO USER PROCESSES
WSR, WORKLOAD 1 OBSERVED
PDP, WORKLOAD 2 OBSERVED
WSR, WORKLOAD 1 SIMULATION
PDP, WORKLOAD 2 SIMULATION

Figure 9.6b

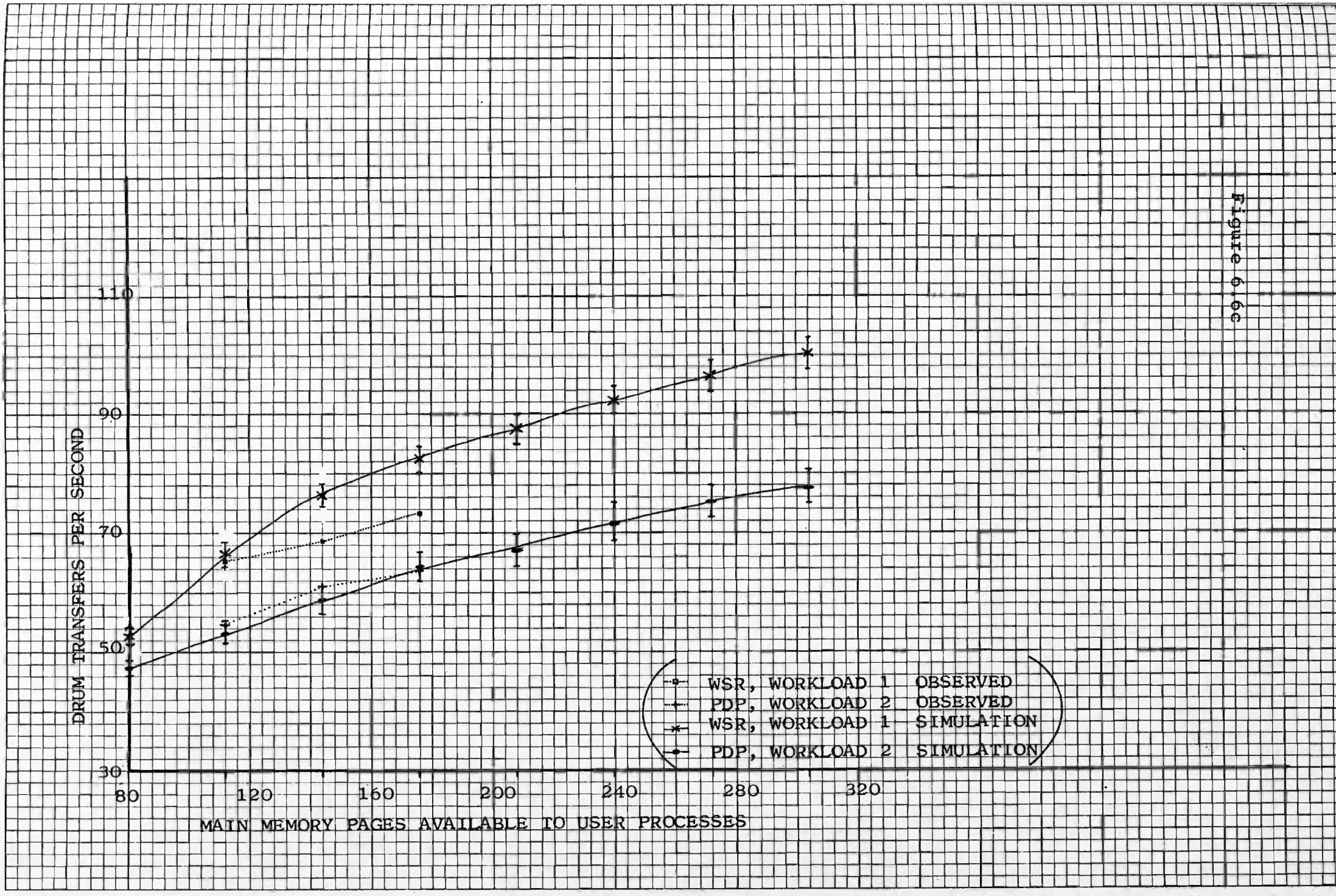

DRUM TRANSFERS PER SECOND
110
90
70
50
30
80
120
160
200
240
280
320
MAIN MEMORY PAGES AVAILABLE TO USER PROCESSES
WSR, WORKLOAD 1 OBSERVED
PDP, WORKLOAD 2 OBSERVED
WSR, WORKLOAD 1 SIMULATION
PDP, WORKLOAD 2 SIMULATION

Figure 6.66

WSR v PDP

The effect of the process loading algorithm upon the system under the two workloads was investigated next. For workload 1, this meant running the system under a pure demand paging scheme and using working set replacement with workload 2. Certain adjustments had to be made to accommodate this. Under both, the ratio of page faults to sleeps in the NEXT PROCESS EVENT function was modified to count all useful preloading transfers as pagefaults - involving increasing the relative number of pagefaults when adjusting from WSR to PDP. Similarly the ratios of different types of pagefaults had to be adjusted. In both cases the fraction of pagefaults involving pages which were already in main memory ('shared' pagefaults) was always kept the same, and the relative numbers of drum and disc faults modified to count all useful preloading transfers as drum pagefaults. Suitable functions were constructed for workload 2 to make the percentage of preloaded pages the same as that observed under WSR on workload 1 (69%, including a preloading wastage set at 25%).

In terms of user CPU (Figure 6.7a) it is noticeable that WSR always gives significantly better performance in lower memory configurations, but that the

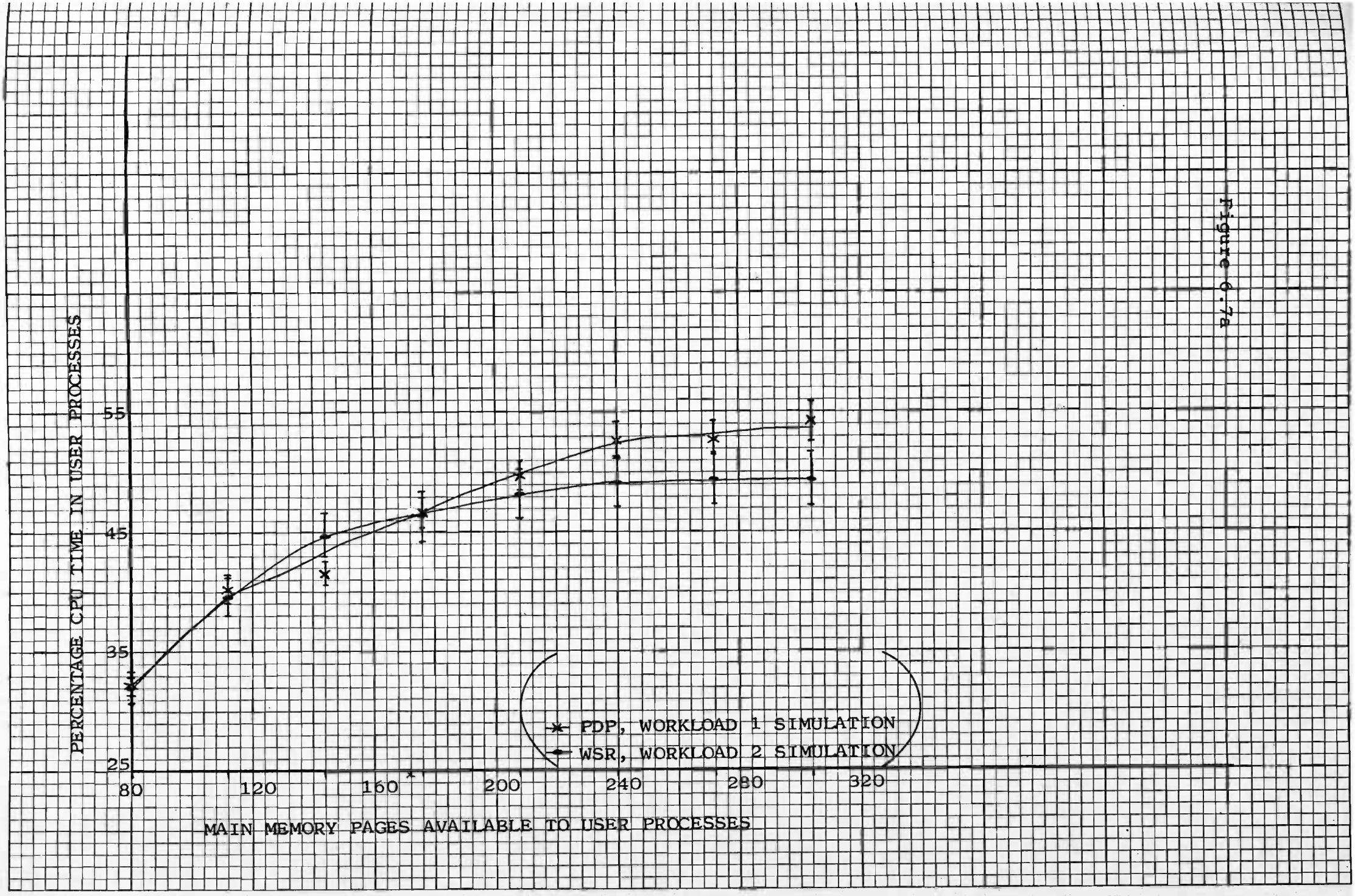

PERCENTAGE CPU TIME IN USER PROCESSES
55
45
35
25
80
120
160
200
240
280
320
MAIN MEMORY PAGES AVAILABLE TO USER PROCESSES
PDP, WORKLOAD 1 SIMULATION
WSR, WORKLOAD 2 SIMULATION

Figure 6.7a

gap between the two narrows as the memory size increases. In the case of workload 2 WSR always gives a better performance, however, for workload 1 WSR gives better performance up to a memory size of 240 pages, then the PDP algorithm appears to do better. The reason for the relatively better performance of PDP in the higher memory sizes may be due to the fact that in these configurations there are larger numbers of processes in main memory, and the number of transfers available when a channel chain is started will also be larger (hence larger channel chains, making more efficient use of the secondary memory). However, for WSR runs on higher memory configurations, with more process swapping being carried out, more and more transfers will be wasted through preloading wastage. The difference between the two workloads (workload 1 processes carry out more work - use more CPU - per residency) tend to indicate that WSR will be more of an advantage when processes use less CPU per entry to main memory and hence cause a higher swap rate. The throughput rate on the drum is always higher under WSR (Figure 6.7b).

Preloading Wastage

The effect of preloading wastage upon the overall system performance is demonstrated by varying this parameter on a model of a configuration providing

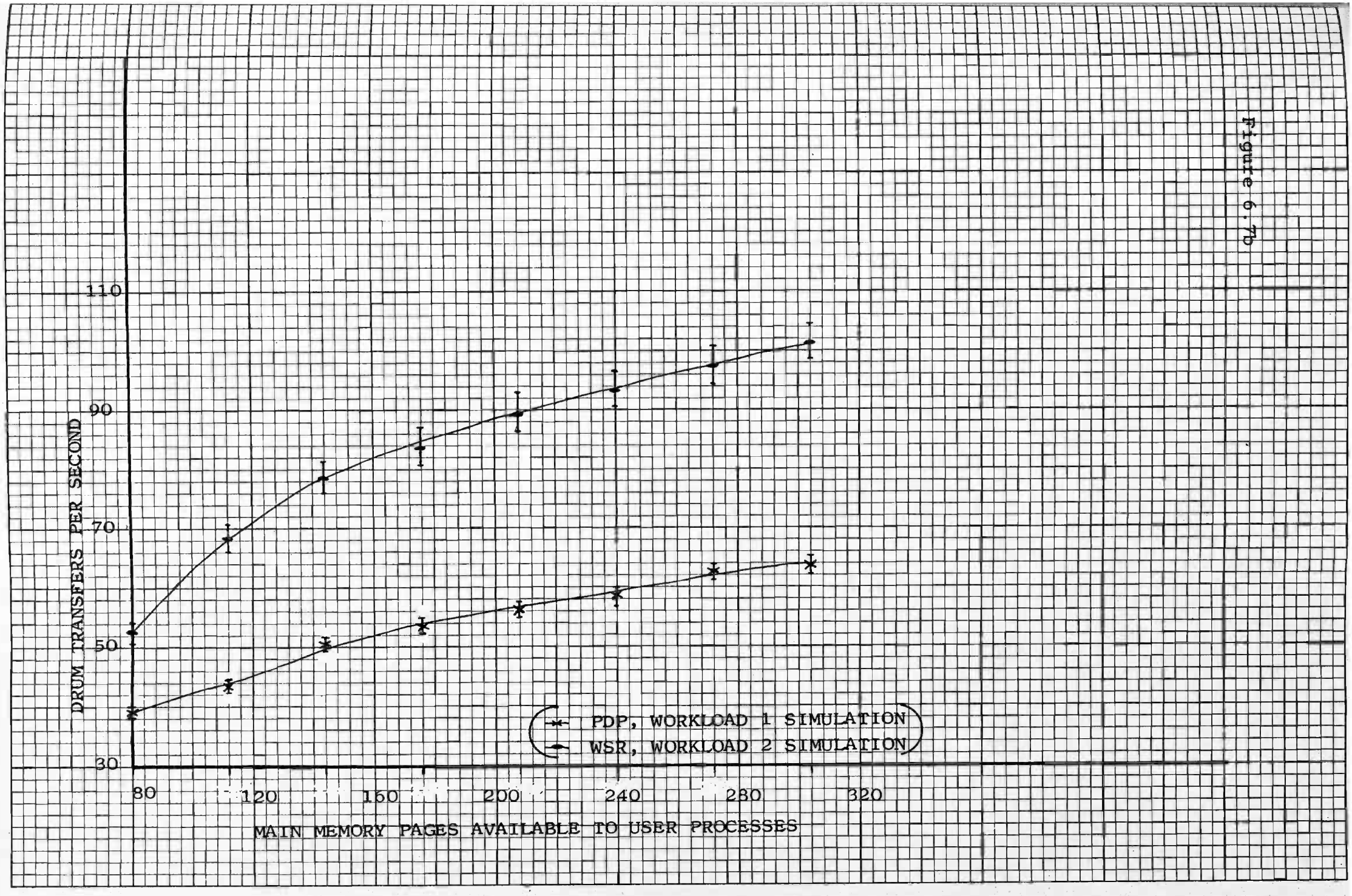

DRUM TRANSFERS PER SECOND
110
90
70
50
30
80
120
160
200
240
280
320
MAIN MEMORY PAGES AVAILABLE TO USER PROCESSES
PDP, WORKLOAD 1 SIMULATION
WSR, WORKLOAD 2 SIMULATION

Figure 6.7b

112 pages to the user processes and using workload 1. The lifetime functions, and all the other process behaviour definition routines, do not take wasted preloaded pages into account, and thus the process behaviour will remain fixed. However, the wasted transfers induced by these pages will interfere with all other processes on the system by soaking up secondary memory bandwidth which may have been put to better use. It may be seen (Figure 6.8a) that even when the wastage reaches 50% the user CPU time is still much greater than that seen with a PDP algorithm. The drum throughput shows a dramatic rise as the wastage falls below 10% - fewer wasted transfers causing a higher swapping rate - and a slight rise when the wastage rises above 30% - with the swapping rate remaining relatively stable but more wasted transfers causing the throughput to rise. (Figure 6.8b).

Secondary Memory Characteristics

The effects of changing certain of the major secondary memory characteristics are next studied using a simulated system of 112 user pages and both workloads - workload 1 using WSR, workload 2 using PDP. The effect of changing the drum latency time - whilst holding the transfer time per page, and all other factors concerning the drum, constant (i.e. 4 sectors) -

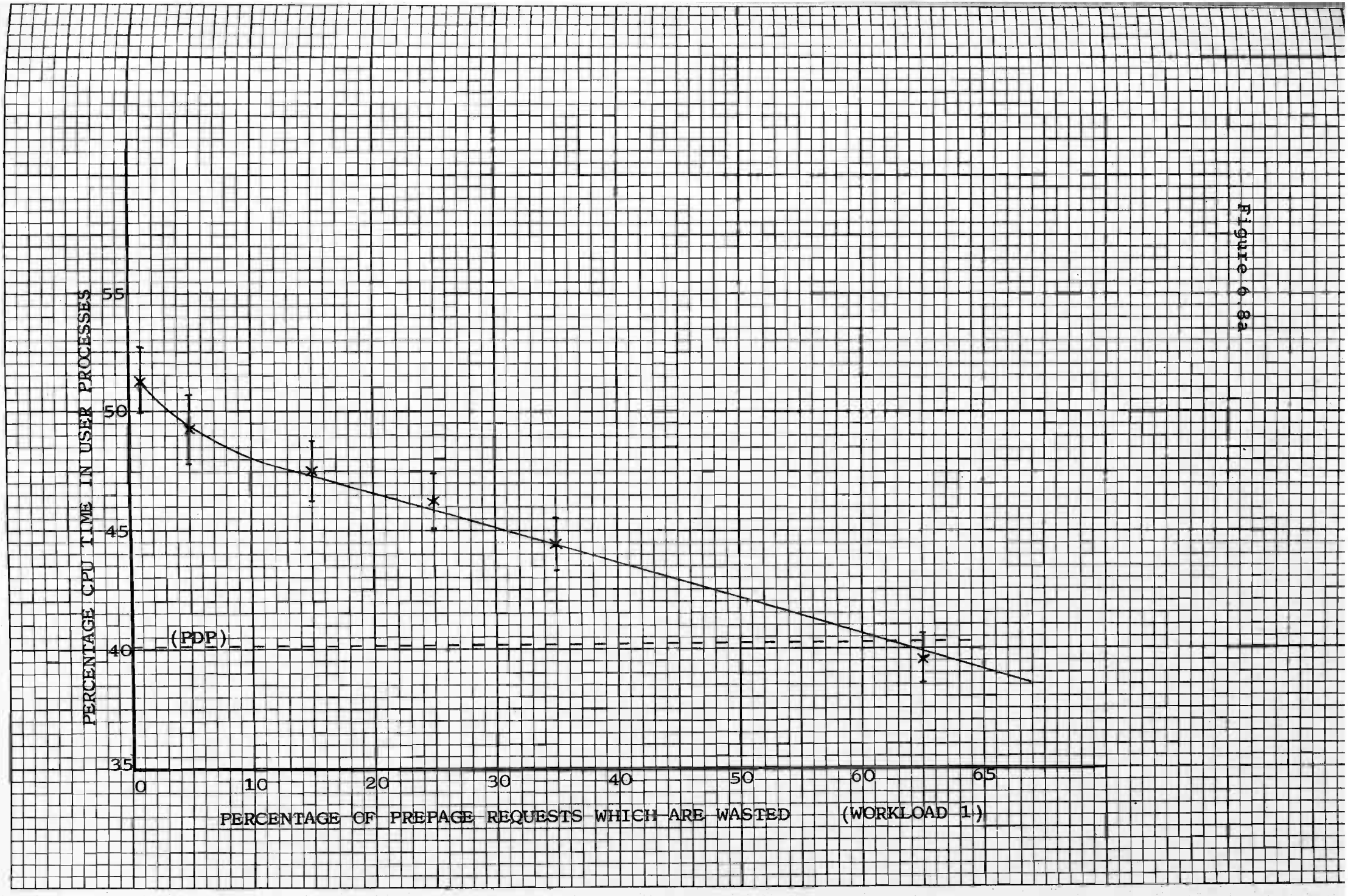

PERCENTAGE CPU TIME IN USER PROCESSES
55
50
45
40
35
(PDP)
0
10
20
30
40
50
60
65
PERCENTAGE OF PREPAGE REQUESTS WHICH ARE WASTED (WORKLOAD 1)

Figure 6.82

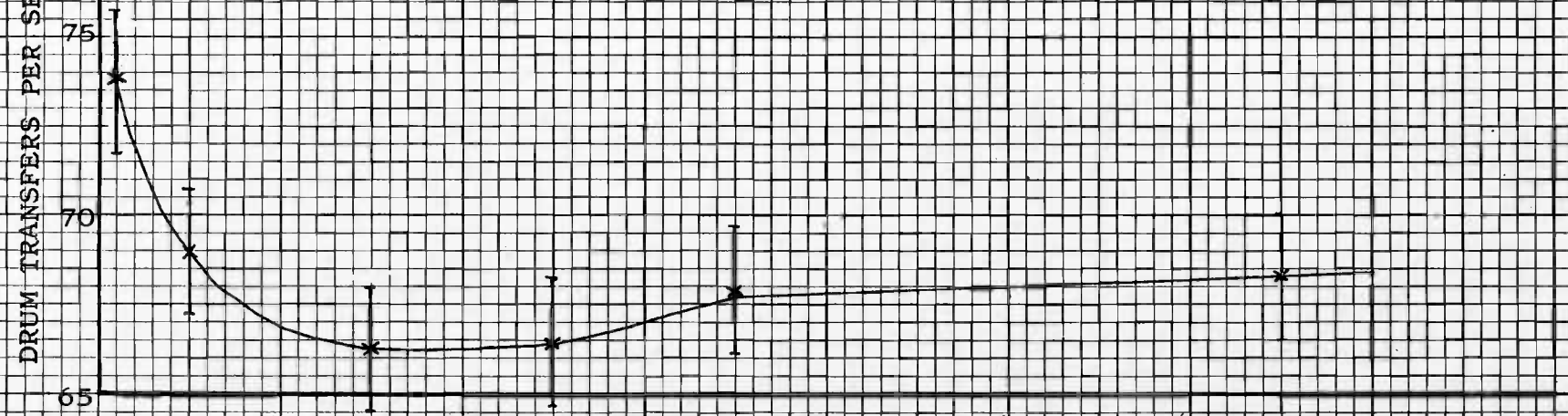

DRUM TRANSFERS PER SECOND
75
70
65
0
10
20
30
40
50
60
PERCENTAGE OF PREPAGE REQUESTS WHICH ARE WASTED (WORKLOAD 1)

Figure 6.8b

is shown first. It may be seen that the user CPU time increases as the latency time decreases (Figure 6.9a) and that the difference between the two workloads increases as the latency increases, the larger latency time affecting the workload with PDP more than that with WSR. Similarly with the drum throughput figures - workload 1 showing a higher drum utilisation than workload 2, even more so with longer latency times. (Fig 6.9b).

The effect of changing the drum transfer time per page (whilst holding all other factors constant) is investigated next. It may be seen that the effect of this upon the two performance metrics is quite dramatic (Figures 6.10a and 6.10b) though the difference between the two workloads remains constant. This indicates that the difference between using a process loading algorithm which involves bulk transfers (e.g. WSR) and one which spaces out these requests through time (as does PDP) will be more dependent upon the secondary memory latency time than the transfer time.

The effect of speeding up the drum (by increasing its rotational speed and assuming that all other elements in the system connected with this - channel-memory bandwidth - will be adequate to cope with this) is studied next. In this, the ratio of transfer time per page to average latency time is held constant

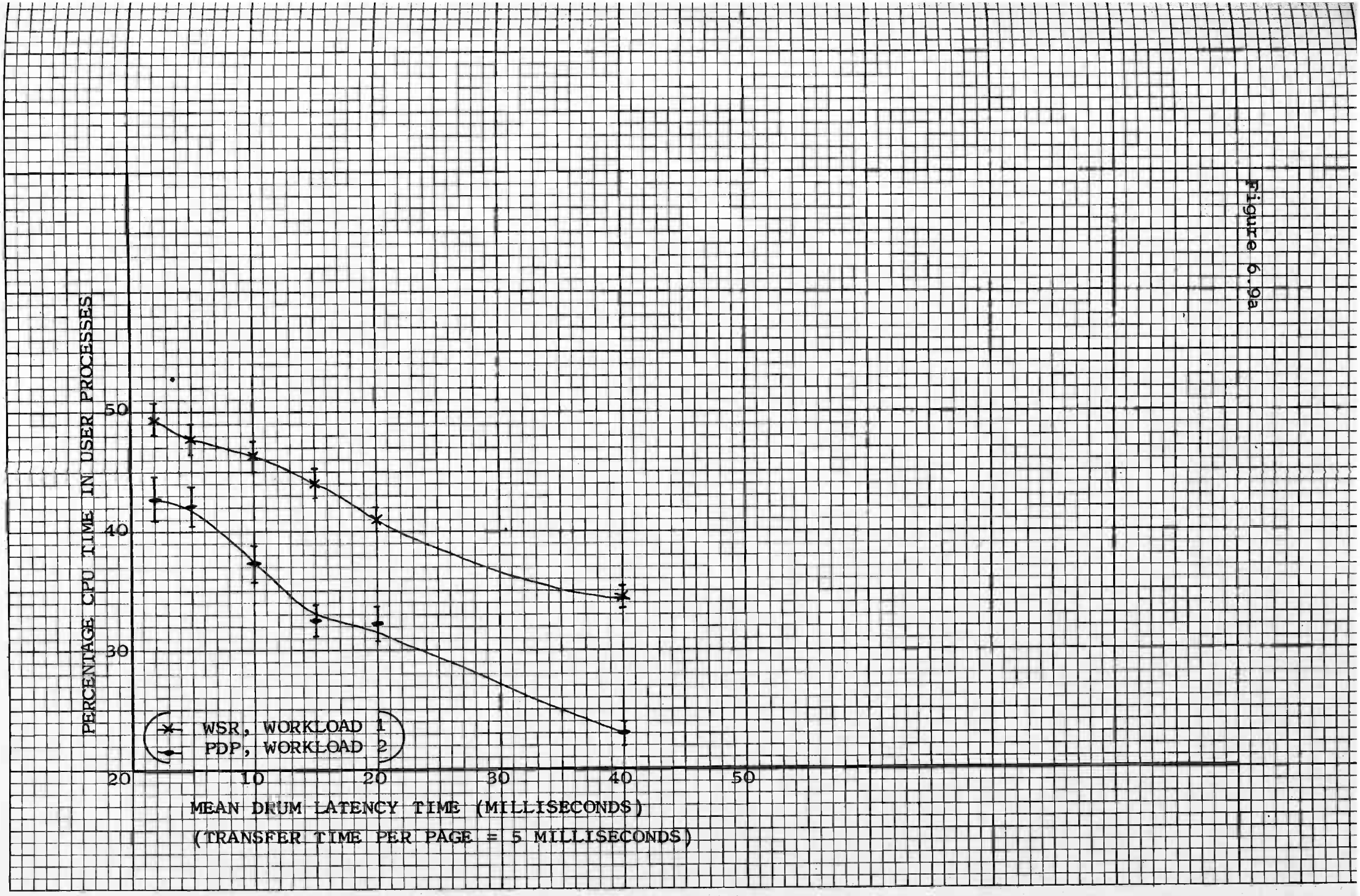

PERCENTAGE CPU TIME IN USER PROCESSES
50
40
30
20
10
20
30
40
50
WSR, WORKLOAD 1
PDP, WORKLOAD 2
MEAN DRUM LATENCY TIME (MILLISECONDS)
(TRANSFER TIME PER PAGE = 5 MILLISECONDS)

Figure 6.9a

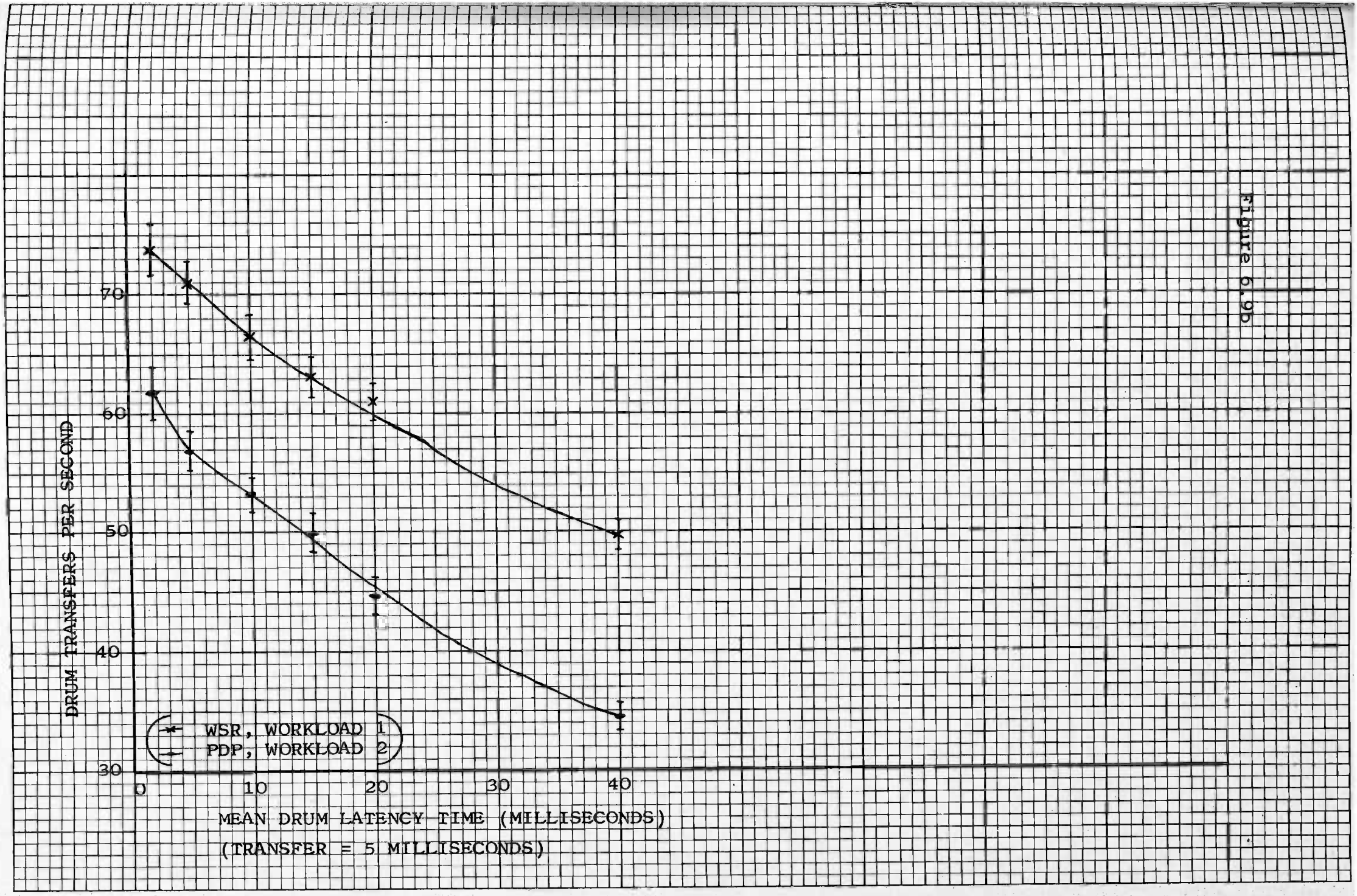

DRUM TRANSFERS PER SECOND
70
60
50
40
30
0
10
20
30
40
WSR, WORKLOAD 1
PDP, WORKLOAD 2
MEAN DRUM LATENCY TIME (MILLISECONDS)
(TRANSFER = 5 MILLISECONDS)

Figure 6.9b

Figure 6.10a

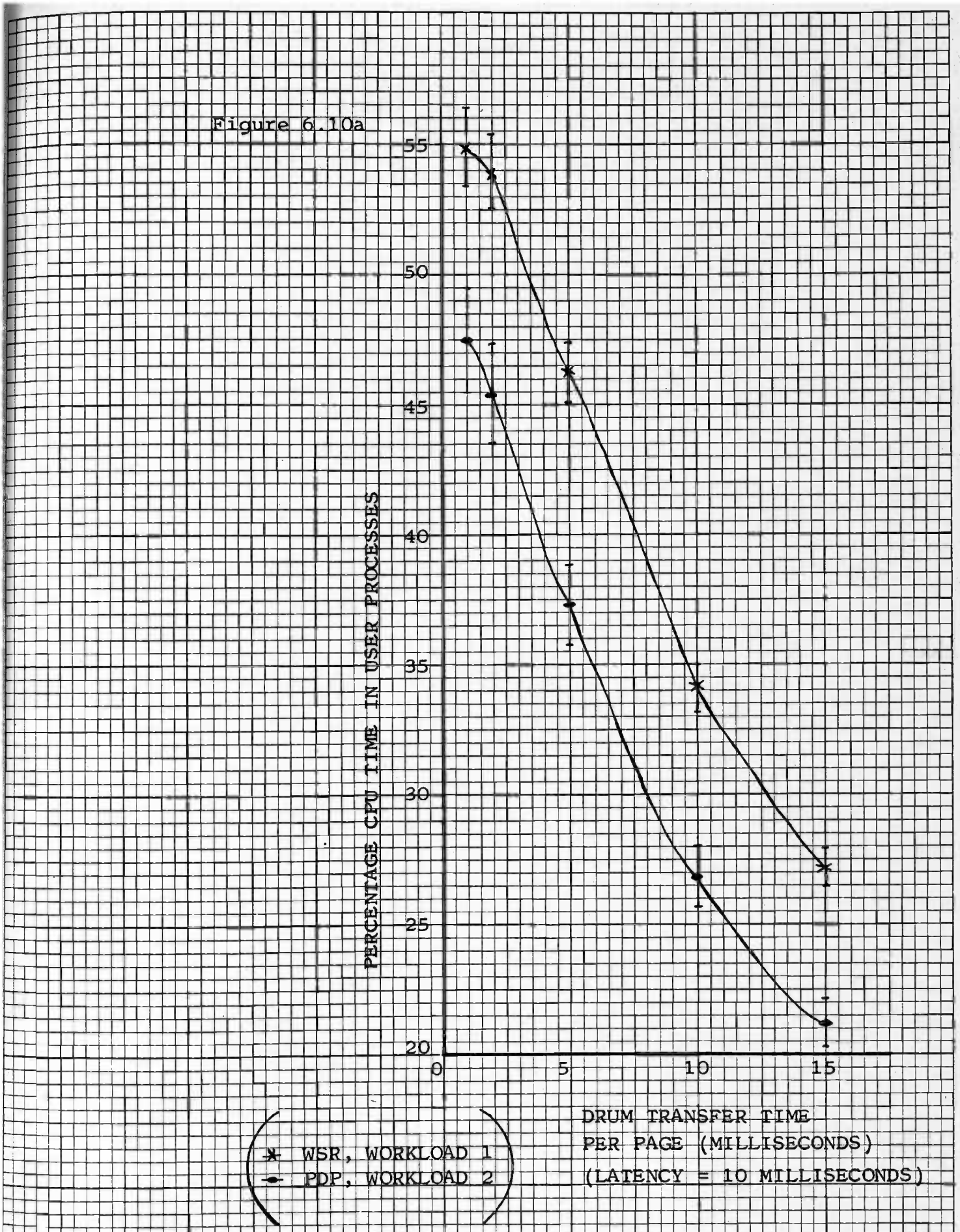
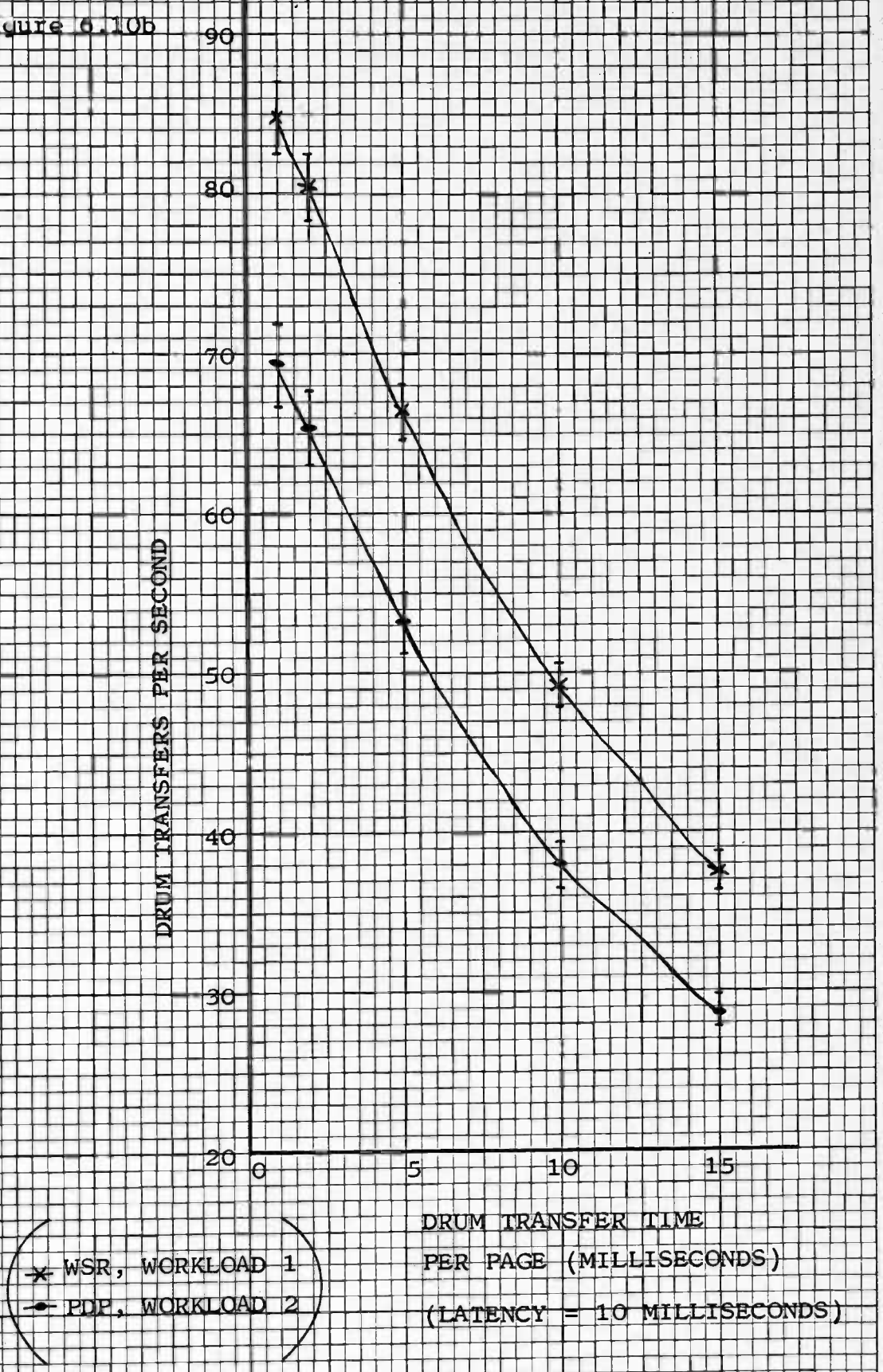

55
50
45
40
35
30
25
20
0
5
10
15
PERCENTAGE CPU TIME IN USER PROCESSES
WSR, WORKLOAD 1
PDP, WORKLOAD 2
DRUM TRANSFER TIME
PER PAGE (MILLISECONDS)
(LATENCY = 10 MILLISECONDS)

Figure 6.10b


90
80
70
60
50
40
30
20
0
5
10
15
DRUM TRANSFERS PER SECOND
DRUM TRANSFER TIME
PER PAGE (MILLISECONDS)
(LATENCY = 10 MILLISECONDS)
WSR, WORKLOAD 1
PDP, WORKLOAD 2

at one to two. As the drum speed increases, the user CPU time continues to rise (Figure 6.11a), until the transfer speed is around one millisecond per page, when this improvement in performance appears to tail off. This is probably because the effect of supervisor overheads begin to dominate at this point (Figure 6.11b) as the drum transfer rate will continue to rise as the drum speed increases (Figure 6.11c). It may also be noted that the supervisor overhead incurred by workload 2 increases much more than that of workload 1 as the drum speed increases.

Intelligent Secondary Memory Channels

The final feature investigated here by the model is that of reducing supervisor overheads in the CPU by putting more processing power into the secondary memory channels. Such an intelligent channel will take care of all sector queuing, the starting of channel chains and the fielding of all interrupts. It would take requests direct from the pagefault handler and the services handling process loading to and unloading from main memory, and send replies to the page-gone and page-here services. The amount of CPU normally accredited to these drum handling services, both from the simulation and observations, is shown in Figure 6.12, and it may be

Figure 6.11a

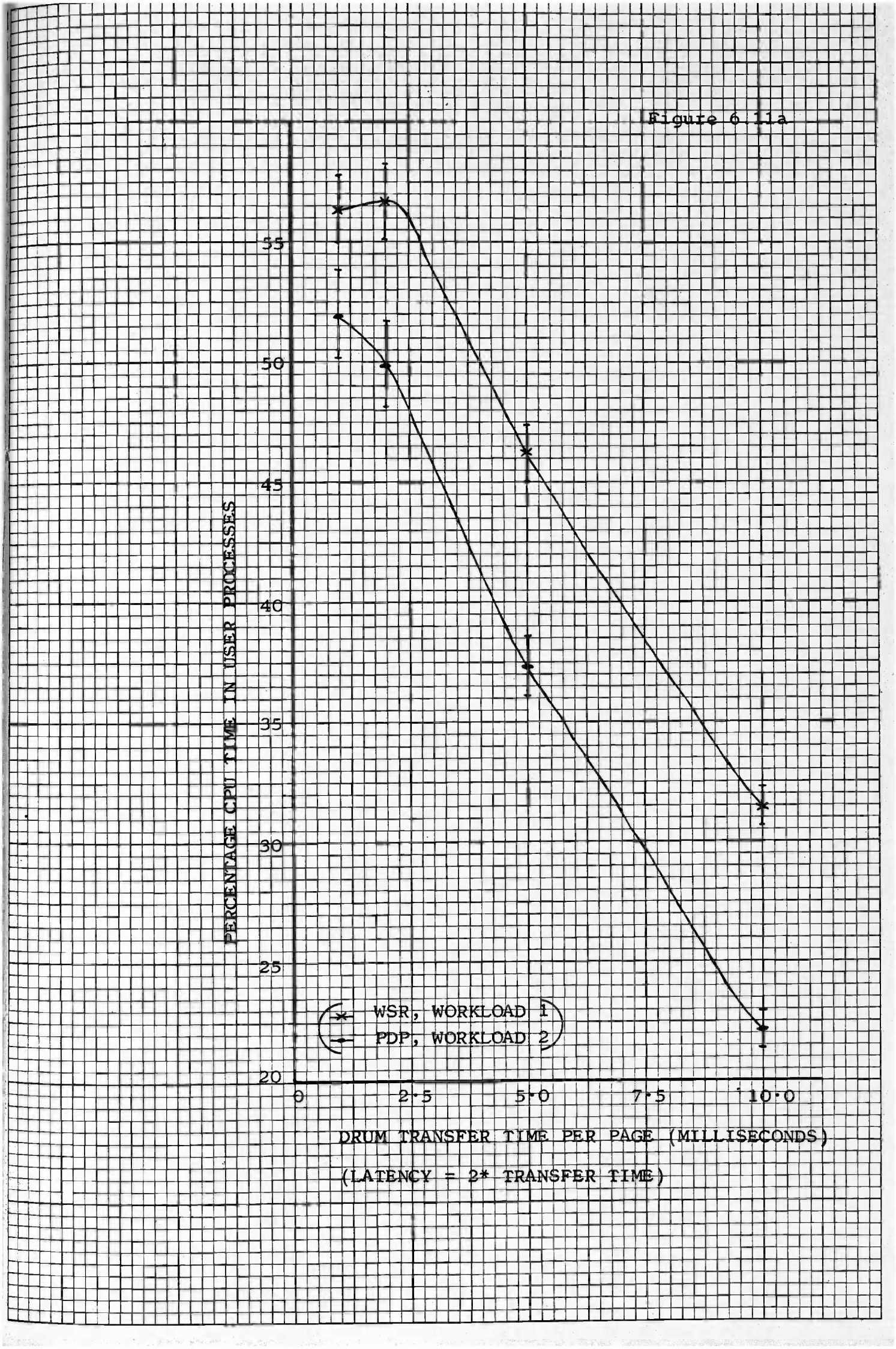

55
50
45
40
35
30
25
20
PERCENTAGE CPU TIME IN USER PROCESSES
WSR, WORKLOAD 1
PDP, WORKLOAD 2
0
2·5
5·0
7·5
10·0
DRUM TRANSFER TIME PER PAGE (MILLISECONDS)
(LATENCY = 2* TRANSFER TIME)

Figure 6.11b

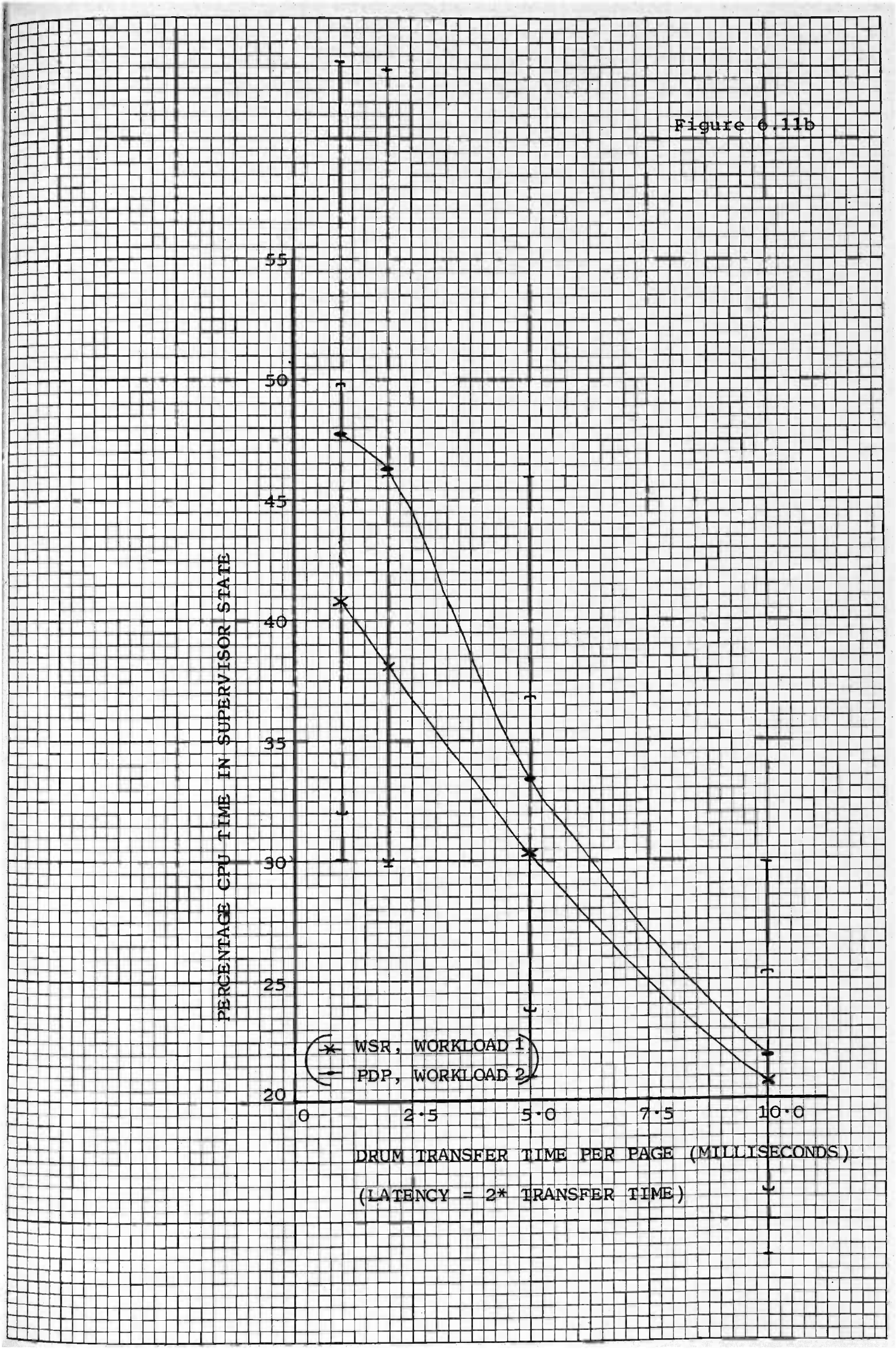

55
50
45
40
35
30
25
20
PERCENTAGE CPU TIME IN SUPERVISOR STATE
WSR, WORKLOAD 1
PDP, WORKLOAD 2
0
2·5
5·0
7·5
10·0
DRUM TRANSFER TIME PER PAGE (MILLISECONDS)
(LATENCY = 2* TRANSFER TIME)

Figure 6.11c

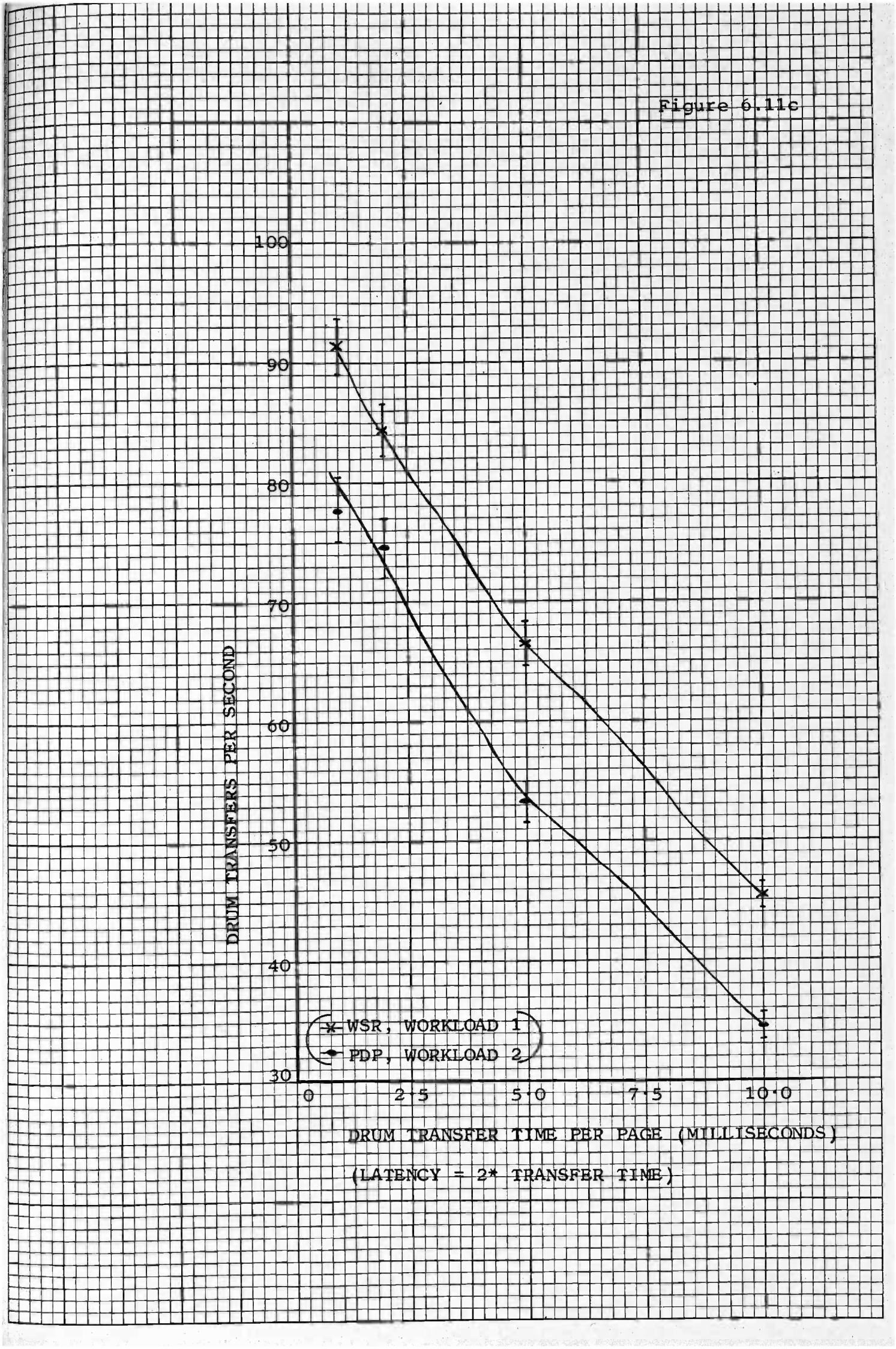
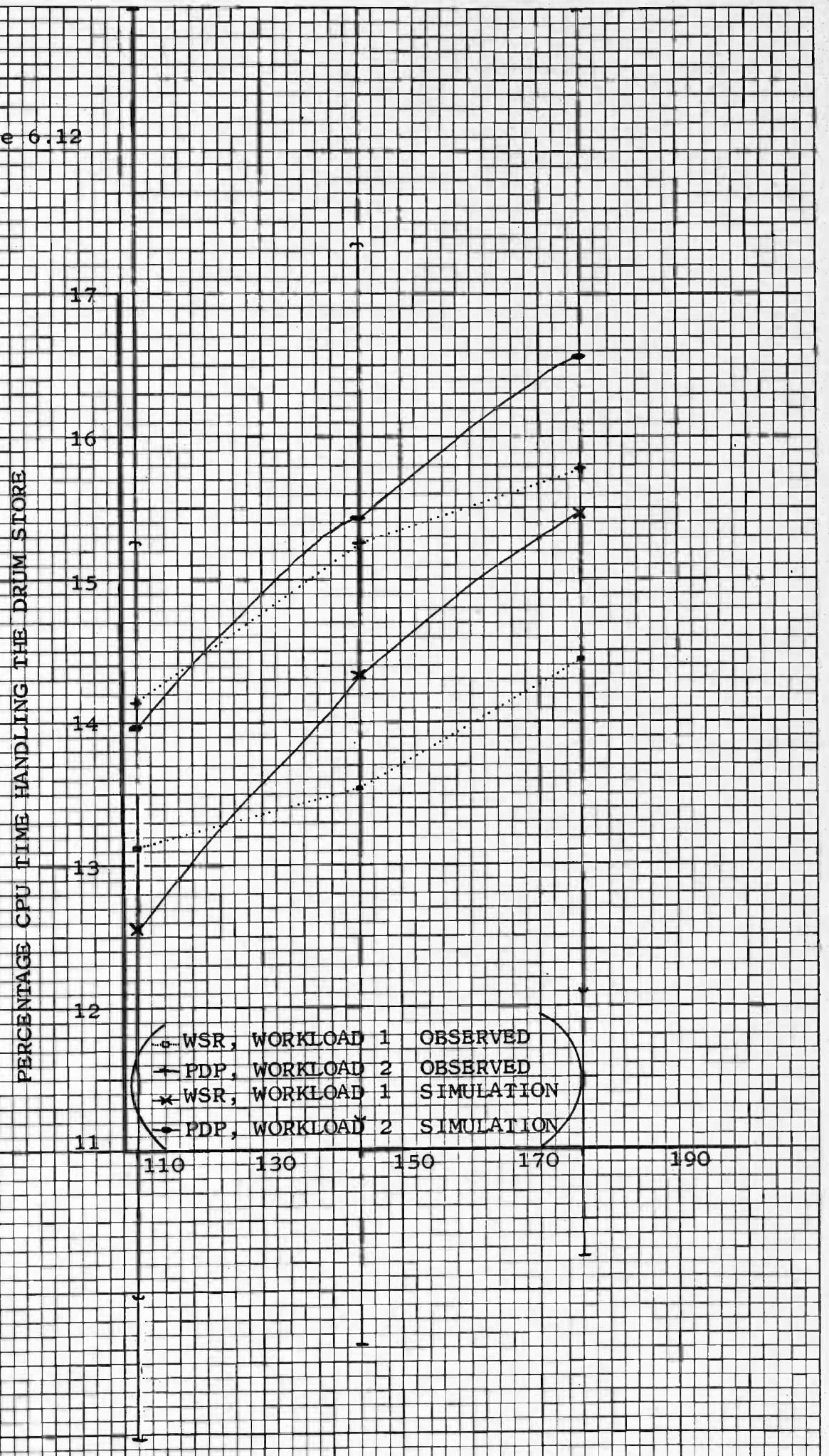

100
90
80
70
60
50
40
30
DRUM TRANSFERS PER SECOND
WSR, WORKLOAD 1
PDP, WORKLOAD 2
0
2·5
5·0
7·5
10·0
DRUM TRANSFER TIME PER PAGE (MILLISECONDS)
(LATENCY = 2* TRANSFER TIME)

Figure 6.12


PERCENTAGE CPU TIME HANDLING THE DRUM STORE
17
16
15
14
13
12
11
WSR, WORKLOAD 1 OBSERVED
PDP, WORKLOAD 2 OBSERVED
WSR, WORKLOAD 1 SIMULATION
PDP, WORKLOAD 2 SIMULATION
110
130
150
170
190
MAIN MEMORY PAGES AVAILABLE TO USER PROCESSES

noted that the simulation always lies within 7·5% of the observed values. The delay times involved in handling these requests in the intelligent channels are held at the same values as were used for the supervisor overheads in the original model, only these operations can now proceed in parallel with other operations on the CPU. The effects of adopting such a scheme show a substantial increase in user CPU time (Figure 6.13a) with the difference between the two workloads again narrowing as the main memory size increases. Similarly the drum throughput is increased (though no changes have been made in the speeds of the drum) and the difference between the drum throughput rates of the standard and intelligent channels increases as the main memory size increases (Figure 6.13b). The rate of increase in supervisor overhead as memory size increases is much less however under the intelligent channel scheme than under the standard one. (Figure 6.13c).

Conclusion

The model presented in this chapter has been used to quantify the effect on overall system performance of changing certain factors which were not, or could not, be included in the EMAS performance experiment. A certain degree of confidence in the

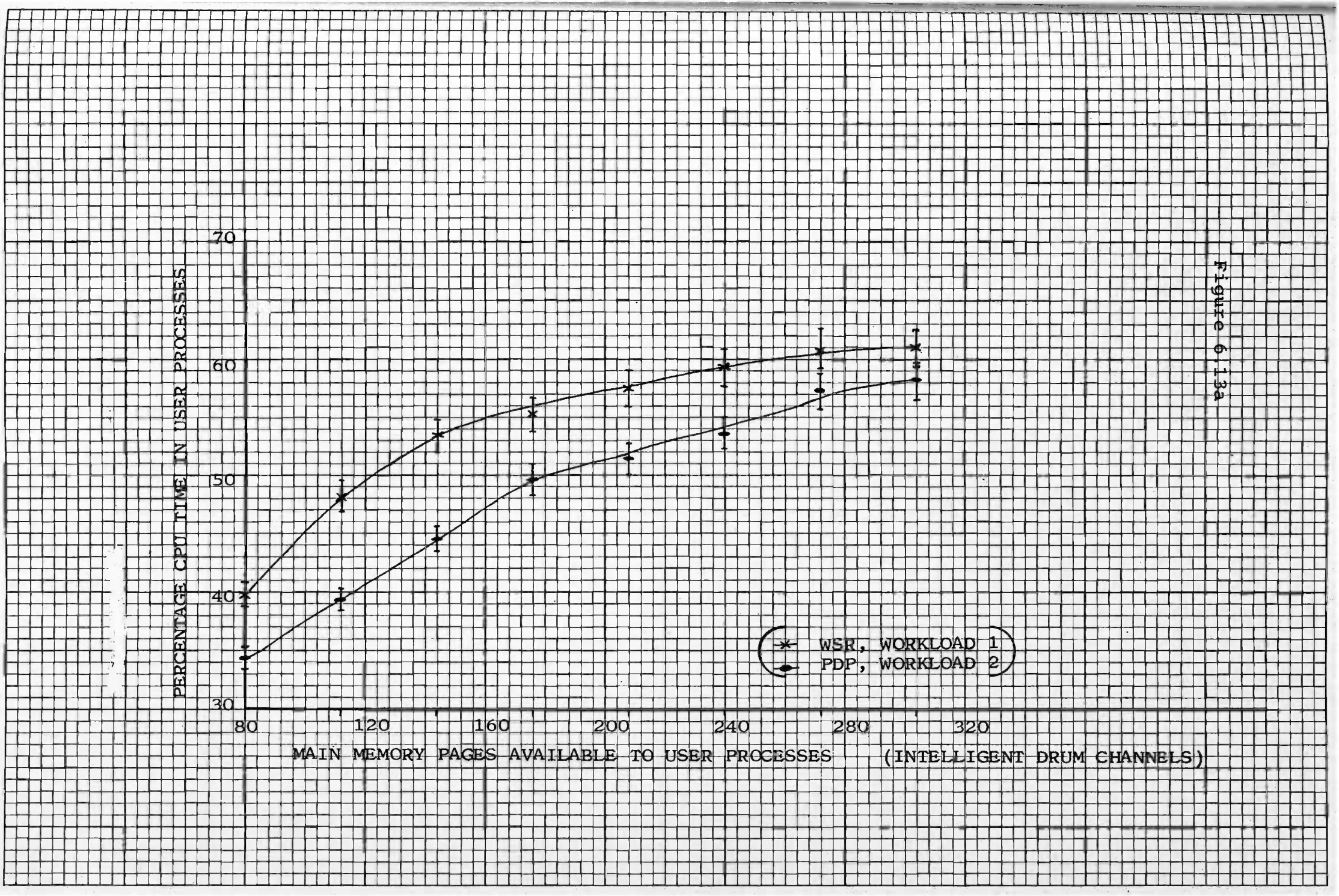

PERCENTAGE CPU TIME IN USER PROCESSES
70
60
50
40
30
80
120
160
200
240
280
320
MAIN MEMORY PAGES AVAILABLE TO USER PROCESSES (INTELLIGENT DRUM CHANNELS)
WSR, WORKLOAD 1
PDP, WORKLOAD 2

Figure 6.13a

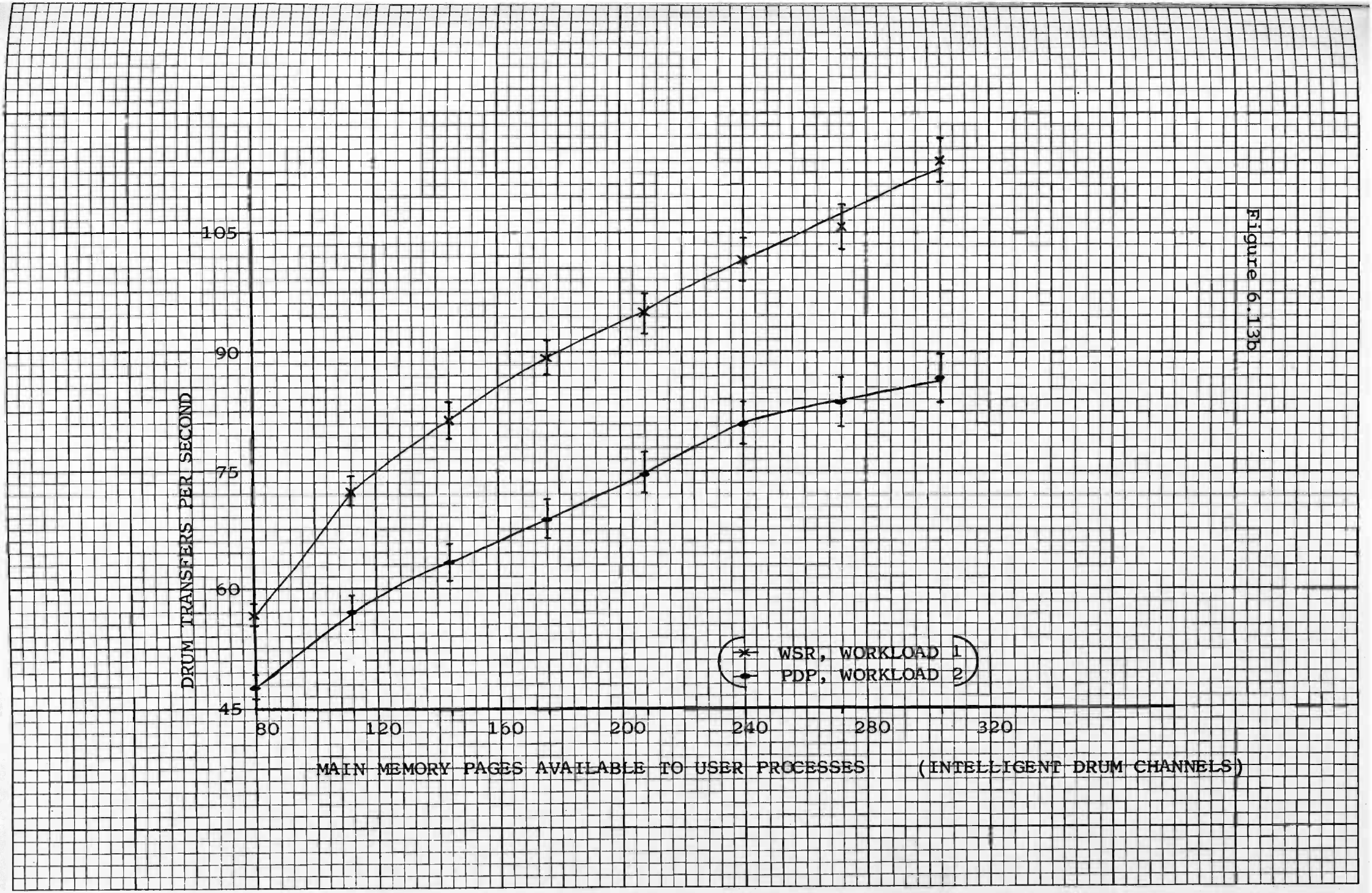

105
90
75
60
45
DRUM TRANSFERS PER SECOND
80
120
160
200
240
280
320
MAIN MEMORY PAGES AVAILABLE TO USER PROCESSES (INTELLIGENT DRUM CHANNELS)
WSR, WORKLOAD 1
PDP, WORKLOAD 2

Figure 6.13b

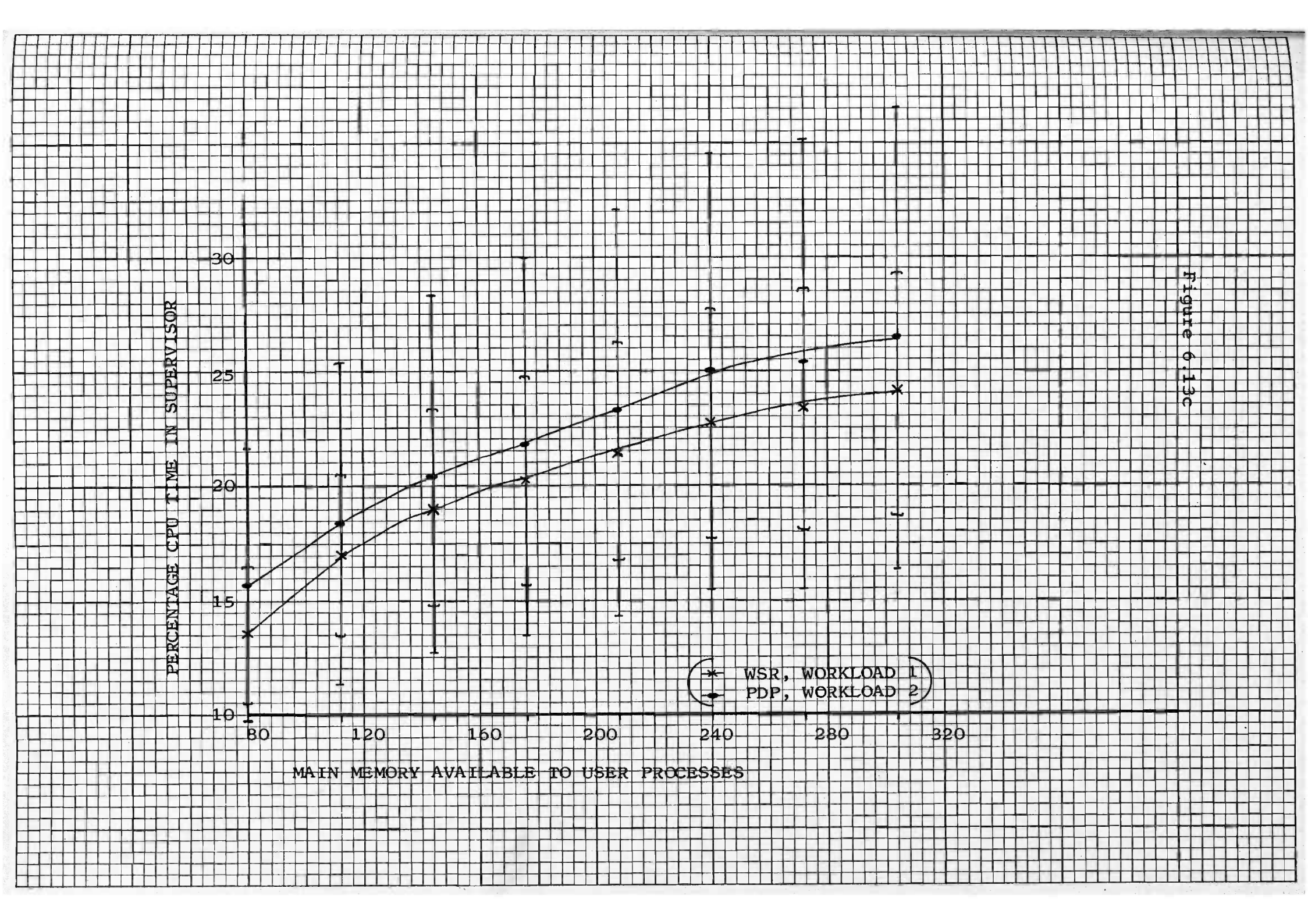

PERCENTAGE CPU TIME IN SUPERVISOR
30
25
20
15
10
80
120
160
200
240
280
320
MAIN MEMORY AVAILABLE TO USER PROCESSES
WSR, WORKLOAD 1
PDP, WORKLOAD 2

Figure 6.13c

model's predictions has been gained through comparing them with empirical observations. The model may be used in yet further investigations of this nature, but it would be better if it were used in conjunction with a further series of empirical experiments (the nature of these experiments may be dictated by results obtained from the model) so that yet more confidence may be gained in the working of the model and its predictions.

Chapter 7

The approach adopted in this work has been to combine the two main techniques available to aid the evaluation of time-shared, virtual memory systems - measurement and modelling. Either of these techniques used singly is subject to certain limitations and it is only by attacking this problem by using the complementary advantages of both techniques that any real progress may be made in this area.

Measurement alone is always limited to the evaluation of existing, functioning systems. In the past this technique has been criticised for producing too much data thus obscuring the relationship between changes in performance and their causes. It is also possible in many cases that the high variability in user workload between two measured intervals may make it impossible to draw any conclusions about the relationships between changes in system structure and system performance. The approach taken here attempts to eliminate this variance in workload by using a remote terminal emulator, and studies the system using a proven experimental design and analysis methods frequently adopted in other branches of experimental science.

The trend towards incorporating mini-computers in the mainframe of the time-shared central processors, with access to most registers and system tables, to handle such tasks as initial programme loading and diagnostics, opens the way to the possibility of using these to monitor system behaviour using hybrid methods - thus hopefully reducing the overhead induced by the measurement process. Similarly the widespread use of mini-computers in the communication networks which service large scale multi-access systems should facilitate the greater use of remote terminal emulation in future measurement experiments.

Modelling is, of course, not restricted to producing results on existing systems but may predict results for any proposed configuration. However, these models will only be of use if they are an accurate reflection of the way in which the system functions. The inability of some modelling techniques to handle real systems has in fact been a matter of some controversy in the literature [Saltzer 1976, Chattergy 1976, Denning 1976]. Where possible a model should be calibrated and validated using measurements taken on a real system. For this to be possible, and for models formulated using a variety of modelling techniques to be tested, a

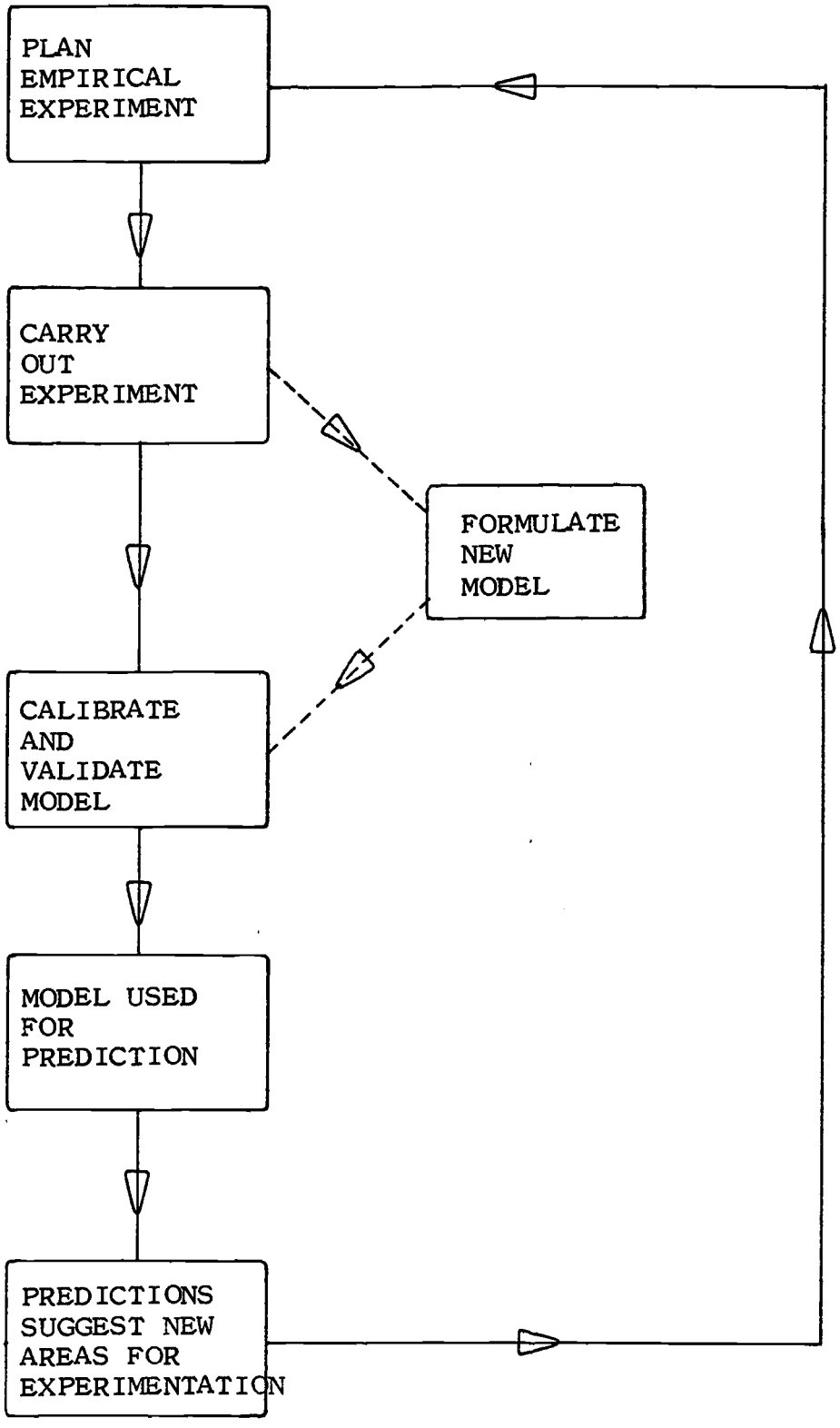
consistent body of measurement data must be available. For this body of data to be consistent it must be obtained under controlled conditions i.e. all parameters (workload as well as system configuration) must be known and able to be reproduced. The empirical techniques presented here provide a method whereby such a body of data may be accumulated.

Not all factors which may impact on the system performance have been covered here - there are too many to enumerate and quantify in such a short time. The effect of varying the number and type of users active on the system is, perhaps, the most notable ommission (though the result of effectively varying the process characteristics is included at two levels). However, there is no reason why this factor should not be studied using the same methodology, and the simulation model modified and extended to include this factor. It must also be noted that the standard workload, used as input to the system during experiments, is itself a model of user behaviour and should ideally be validated against measurements of real user behaviour before being put to such a use.

The ideal way therefore for any process of system evaluation on an existing system

to proceed is for a model of system behaviour to be initially derived from, and to be validated with, measurements from a set of controlled experiments (Figure 7.1). Results from this model may then be used to suggest new areas for experimentation and the model may be further validated by results produced by these. It is only by carrying out such an exercise that any confidence can be gained in a modelling technique and its applicability to complex systems. These proven modelling techniques may then be used with slightly more confidence in predicting the behaviour of completely new systems and will hopefully be validated by a similar empirical experimental programme when such systems are built.

Figure 7.1


PLAN EMPIRICAL EXPERIMENT
CARRY OUT EXPERIMENT
FORMULATE NEW MODEL
CALIBRATE AND VALIDATE MODEL
MODEL USED FOR PREDICTION
PREDICTIONS SUGGEST NEW AREAS FOR EXPERIMENTATION

Appendix

Analysis of Variance

The experimental design adopted in the EMAS performance experiment is a full 3 x 2 x 2 factorial experiment [Cochran and Cox 1957] in which the following factors are varied:

FACTOR	NUMBER OF LEVELS
A - Main Memory Size	3
B - Secondary Memory Channels	2
C - Scheduling Algorithm	2

The results from each run in this experiment are considered to be in the form

$$y = M + \underbrace{\chi_1 a + \chi_2 a^2 + \chi_3 b + \chi_4 c}_{\text{MAIN EFFECTS}}$$

$$+ \underbrace{\chi_5 ab + \chi_6 a^2 b + \chi_7 ac + \chi_8 a^2 c + \chi_9 bc}_{\text{SECOND ORDER EFFECTS}}$$

$$+ \underbrace{\chi_{10} abc + \chi_{11} a^2 bc}_{\text{THIRD ORDER EFFECTS}} + E$$

Where:

M - the overall mean - all factors present at level 1.

E - an error term due to random experimental errors (and sometimes environmental effects - assumed to have been eliminated in this experiment).

χ coefficients - variance around the mean due to the factors included in the experiment and interactions between those factors (second and third order effects).

a,b,c - experiment run having the main factors (A, B or C respectively) present at level 2 (rather than level 1).

a^2 - experiment run having the main factor A present at level 3.

The analysis of variance (ANOVA) technique [Yates 1937, Johnstone and Leone 1964, Mendenhall 1968] merely determines the values of the χ coefficients and thus quantifies the effect of each of the factors and their interactions. It is normal

to deduce the error term from duplication of certain or all of the runs. However since it was not possible to carry out any duplication the approach adopted [Mendenhall 1968] was to assume that the higher order effects are negligible and that the effect attributed to these may be used as an estimate of the experimental error. The ratio of the sum of squares of each main effect to the sum of squares of this error term is then used in conjunction with an F - test to decide whether or not the effect of each factor is statistically significant, and if so then to what degree.

Bibliography

List of abbreviations:

ACM	-	Association for Computing Machinery
AFIPS	-	American Federation of Information Processing Societies
CACM	-	Communications of the ACM
CMG	-	Computer Measurement Group
ERCC	-	Edinburgh Regional Computing Centre
FJCC	-	Fall Joint Computer Centre
IBM	-	International Business Machines
ICS	-	International Computing Symposium
IRE	-	Institute of Radio Engineers
IRIA	-	Institut de Recherche d'Informatique et d'Automatique
IUCC	-	Inter Universities Computer Conference
JACM	-	Journal of the ACM
MIT	-	Massachusetts Institute of Technology
NCC	-	National Computer Conference
PER	-	Performance Evaluation Review
SIGOPS	-	ACM Special Interest Group on Operating Systems
SJCC	-	Spring Joint Computer Conference
US-NBS	-	United States of America National Bureau of Standards

Abrams, M.D. and Cotton, I.W.
The Service Concept Applied to Computer Networks
US-NBS technical note 880. 1975.

Abrams, M.D., Treu, S. and Blanc, R.P.
Measurement of Computer Communications Networks
US-NBS technical note 908. 1976.

Adams, J.C., Gelenbe, E. and Vicard, J.
An Experimentally Validated Model of the Paging Drum
Proc. Sigmetrics - CMG VIII, Washington. Nov. 1977.

Adams, J.C. and Millard, G.E.
Performance Measurements on the Edinburgh Multi-Access System
Proc. ICS - 75, Antibes, France. 1975.

Adams, J.C., Currie, W.S. and Gilmore, B.A.C.
The Structure and Uses of the Edinburgh Remote Terminal Emulator
Presented at IUCC 1977 (at University of East Anglia).
University of Edinburgh, Department of Computer Science Internal Report CSR-12-77.

Alexander, M.T.
Organisation and Features of the Michigan Terminal System
AFIPS, FJCC vol. 41. 1972.

Alexander, M.T.
The MTS Data Collection Facility
University of Michigan Computing Centre Memo M294. April 1975.

Aschenbrenner, R.A., Annist, L. and Natarjan, N.K.
The Neutron Monitor System
AFIPS, FJCC vol. 39. 1971.

Badel, M. and Zonzon, M.
Validation d'un Modele a Processus de Diffusion pour un Reseau de Files d'Attente General: Application a Cyclades
IRIA research report 209. Dec. 1976.

Barber, E., Asphyell, A. and Dispen, A.
Benchmark Construction
Sigmetrics PER vol. 4 no. 4. 1975

Bard, Y.
An Analytical Model of CP-67 and VM/370
in Computer Architecture and Networks, Gelenbe and Mahl, R. Editors. 1975.

Bard, Y.
A Characterisation of VM/370 Workloads in Modelling and Performance Evaluation of Computer Systems, Beilner, H. and Gelenbe, E. Editors. 1977.

Bard, Y.
Conversations with the author. April 1977.

Bard, Y.
Experimental Evaluation of System Performance
IBM Systems Journal vol. 12 no. 3. 1973.

Bard, Y.
Performance Criteria and Measurement for a Time-Sharing System
IBM Systems Journal vol. 10 no. 3. 1971.

Baskett, F., Chandy, M., Muntz, R. and Palacious, J.
Open, Closed and Mixed Networks of Queues with Different Classes of Customers
JACM vol. 22. 1975.

Baudet, G., Boulender, J. and Ferrie, J.
Analysis of a Drum with Bulk Arrivals
Proc. ICS 1975, Antibes, France.

Belady, L.A. and Keuhner, C.J.
Dynamic Space Sharing in Computer Systems
CACM vol. 12 no. 5. 1969.

Bobrow, Burchfiel, Murphy and Tomlinson
Tenex - A Paged, Time Sharing System for the PDP-10
CACM vol. 15 no. 3. 1972.

Boehm, B.W., Seven, M.J. and Watson, R.A.
Interactive Problem Solving - An Experimental Study of Lockout Effects
AFIPS SJCC. 1971.

Bryan, G.E.
Joss 20,000 Hours at a Console, a Statistical Summary
AFIPS, FJCC vol. 31. 1967.

Buchanan, I. and Duce, D.A.
An Interactive Benchmark for a Multi-User Mini-computer System
ACM Sigmetrics PER vol. 5 no. 4. 1976.

Buzen, J.P.
Computational Algorithms for Closed Queuing Networks with Exponential Servers
CACM vol. 16. 1973.

Callaway, P.H.
Performance Measurement Tools for VM/370
IBM Systems Journal vol. 14 no. 2. 1975.

Chang, W.
Single Server Queuing Processes in Computing Systems
IBM Systems Journal vol. 9 no. 1. 1970.

Chattergy, R.
Memory Management Modelling
CACM Forum vol. 19 no. 8. 1976.

Cochran, W.G. and Cox, G.M.
Experimental Designs
Pub. Wiley. 1957.

Corbato, F.J., Merwin-Dogget, M. and Daley, R.C.
An Experimental Time Sharing System
AFIPS, SJCC vol. 21. 1962.

Corbato, F.J., Saltzer, J.H. and Clingen, C.T.
MULTICS - the First Seven Years
AFIPS, SJCC vol. 40. 1972.

Corbato, F.J. and Vyssotsky, J.A.
Introduction and Overview of the MULTICS System
AFIPS, FJCC vol. 27. 1965.

Crissman, P.A.
The Compatible Time Sharing System: A Programmes Guide
MIT press. 1965.

Dahl, O.J. and Nygaard, K.
Simula - an Algol Based Simulation Language
CACM vol. 9 no. 9. 1966.

Daley, R.C. and Neumann, P.G.
A general Purpose File System for Secondary Storage
AFIPS, FJCC vol. 27. 1965.

David, E.E. and Fano, R.M.
Some Thoughts about the Social Implications of Accessible Computing
AFIPS, FJCC vol. 27. 1965.

Denning, P.J.
Memory Management Modelling
CACM Forum vol. 19 no. 8. 1976.

Denning, P.J.
The Working Set Model for Program Behaviour
CACM vol. 11 no. 5. 1968.

Denning, P.J.
Thrashing: Its Causes and Prevention
AFIPS, FJCC vol. 33. 1968.

Denning, P.J.
Virtual Memory
Computing Surveys vol. 12 no. 3. 1970.

Dennis, J.B.
Segmentation and the Design of Multiprogrammed Computer Systems
JACM vol. 12 no. 4. 1965.

Digital Equipment Corporation
DEC System 10 Reference Manual
DEC - 10 - HGAD - D. 1972.

Digital Equipment Corporation
Large Computer News vol. 1 no. 8. 1977.
Pub. by Digital Equipment Corporation.

Dimsdale, B. and Markowitz, H.M.
A Description of the Simscript Language
IBM Systems Journal vol. 3 no. 1. 1964.

Emery, A.R. and Alexander, M.T.
A Performance Comparison of the Amdahl 470 V/6 and the IBM 370/168
University of Michigan Computing Centre. 1975.

Estrin, G., Muntz, R.R. and Uzgalis, R.C.
Modeling, Measurement and Computer Power
AFIPS, SJCC vol. 40. 1972.

Fryer, R.E.
The Memory Bus Monitor - A New Device for Developing Real Time Systems
Proc. NCC vol. 42. 1973.

Gelenbe, E.
On Approximate Computer System Models
JACM vol. 22. 1975.

Gelenbe, E. and Muntz, R.R.
Probabilistic Models of Computer Systems - Part 1
Acta Informatica vol. 7. 1976.

Gibson, J.C.
The Gibson Mix
IBM internal publication TR 00 2043. 1970.

Gilmore, B. and McBride, B.
Stimulator Documentation
ERCC. 1975.

Glasser, E.L., Couleur, J.F. and Oliver, G.A.
System Design of a Computer for Time-Sharing Applications
AFIPS, FJCC vol. 27. 1965.

Gold, M.M.
Time Sharing and Batch Processing: An Experimental Comparison of Their Values in a Problem Solving Situation
CACM vol. 12 no. 5. 1969.

Gonzales, C.M.
Performance Measurements of the Scheduler in the PDP-10 TENEX Operating System
Case Western Reserve University, Ph.D. Thesis. Jan. 1975.

Gordon, W.J. and Newell, G.F.
Closed Queuing Systems with Exponential Servers
Operational Research vol. 15 no. 2. 1967.

Greenbaum, H.J.
A simulator of Multiple Interactive Users to Drive a Time Shared Computer System
Thesis MIT project MAC 1969 MAC-TR-54.

Grenader, U. and Tsao, R.F.
Quantitative Methods for Evaluating Computer System Performance: A Review and Proposals
Statistical Computer Performance Evaluation. Academic Press, Freiberger, W. Editor. 1972.

Holdsworth, D., Robinson, G.W. and Wells, M.
A Multi Terminal Benchmark
Software Practice and Experience vol. 3 no. 1. 1973.

IBM Data Processing Division
CP-67/CMS Version 3 System Description Manual
IBM, White Plains, N.Y., Form GH20-0802-1. 1970.

IBM Data Processing Division
IBM Virtual Machine Facility/370
IBM, White Plains, N.Y., Form GC20-1800. 1972.

Jackson, J.R.
Jobshop-like Queuing Systems
Management Science vol. 10. 1963.

Jalics, P.J.
Measurements of the PDP-10 TOPS-10 Operating System
Case Western Reserve University, Ph.D. Thesis.
Jan. 1973.

Johnston, N.L. and Leone, F.C.
Statistics and Experimental Design in Engineering and Physical Sciences. Volumes I and II

Pub. Wiley. 1964.

Kilburn, T., Edwards, D.B.G., Lanigan, M.J. and Sumner, F.H.
One-Level Storage System
IRE transactions on electronic computers, EC-11 no.2. 1962.

Kosko, D. and Turner, R.
A Report on a Data Transaction Processing Experiment
DEC internal report, 75 DK 378-440. Nov. 1975.

Lassettre, E.R. and Scherr, A.L.
Modelling the Performance of the OS/360 Time Sharing Option
in Statistical Computer Performance Evaluation,
Academic Press, Freiberger, W. Editor. 1972.

Lehmann, M.M. and Gomma, H.
Interactive System Performance in a Simulated Environment
Imperial College London, Department of Computing and Control Research, report 73/9. 1973.

Leruodier, J. and Parent, M.
Discrete Event Simulation Modelling of Computer Systems for Performance Evaluation
IRIA research report no. 177. 1976.

Lindsay, D.S.
A Hardware Monitor Study of a CDC KRONOS System
Proc. International Symposium on Computer Performance, Modelling, Measurement and Evaluation, Harvard. 1976.

Lucas, H.C.
Performance Evaluation and Monitoring
Computing Surveys vol. 3 no. 3. 1971.

Lynch, W.
Letter to the Author. 1975.

McKinney, J.M.
A Survey of Analytical Time Sharing Models
Computing Surveys vol. 1 no. 2. 1969.

Mendenhall, W.
The Design and Analysis of Experiments
Pub. Wadsworth. 1968.

Merrill, H.W.B.
Further Comments on Comparative Evaluation of Kiviat Graphs
Sigmetrics PER vol. 4 no. 1. 1975.

Meyer, R.A. and Seawright, L.H.
A virtual Machine Time Sharing System
IBM Systems Journal vol. 9 no. 3. 1970.

Millard, G.E.
Conversations with the author. 1975.

Millard, G.E., Rees, D.J. and Whitfield, H.
The Standard EMAS Subsystem
Computer Journal vol. 18 no. 3. 1975.

Miller, R.B.
Response Time in Man-Computer Conversational Transactions
AFIPS, FJCC vol. 33. 1968.

Moore, C.G.
Network Models for Large Scale Time-Sharing Systems
University of Michigan, Ann Arbor. Ph.D. Thesis. 1971.

Murphy
Storage Organisation and Management in TENEX
AFIPS, FJCC vol. 41. 1972.

Nutt, G.J.
Tutorial: Computer System Monitors
Computer. Nov. 1975.

Organic, E.I.
The MULTICS System: An Examination of its Structure
MIT Press. 1972.

Ossanna, J.F., Mikus, L.E. and Dunten, S.D.
Communication and Input/Output Switching in a Multiplexed Computing System
AFIPS, FJCC vol. 27. 1965.

Partridge, D.R. and Card, R.E.
Hardware Monitoring of Real Time Aerospace Computer Systems
Proc. International Symposium on Computer Performance, Modelling, Measurement and Evaluation, Harvard. 1976.

Pinkerton, T.B.
Performance Measurement in a Time Sharing System
CACM vol. 12 no. 11. 1969.

Potier, D.
An Analysis of Prepaging Policies
Proc. ACM SIGOPS Conference, Purdue. 1977.

Rees, D.J.
The EMAS Director
Computer Journal vol. 18 no. 2. 1975.

Ruud, R.J.
The CPM-X Systems Approach to Performance Measurement
AFIPS, FJCC vol. 41 pt. II. 1972.

Saltzer, J.H.
On the Modelling of Paging Algorithms
CACM Forum vol. 19 no. 5. 1976.

Saltzer, J.H. and Gintell, W.
The Instrumentation of MULTICS
CACM vol. 13 no. 8. 1970.

Scherr, A.L.
An Analysis of Time-Shared Computer Systems
Project MAC, MIT Ph.D. Thesis, MAC TR-18. 1965.

Schreiber, H.
Hardware Measurement of CPU Activities
Modelling and Performance Evaluation of Computer Systems, Beilner, H. and Gelenbe, E. Editors. 1977.

Schroeder, M.D.
Performance of the GE-645 Associated Memory While MULTICS is in Operation
Proc. Symposium on System Performance Evaluation, Harvard. 1971.

Schwemm, R.E.
Experience Gained in the Development and Use of TSS
AFIPS, SJCC vol. 40. 1972.

Sekino, A.
Performance Evaluation of Multiprogrammed Time-Shared Computer Systems
Project MAC, MIT Ph.D. Thesis, MAC TR-103. 1972.

Shelness, N.H., Stephens, P.D. and Whitfield, H.
The Edinburgh Multi Access System, Scheduling and Allocation Procedures in the Resident Supervisor
Springer Verlag lecture notes in Computer Science no. 19. 1974.

Stang, H. and Southgate, P.
Performance Evaluation of 3rd Generation Computing Systems
Datamation vol. 15. 1969.

Stasuik, J.
Terminal Driver Monitor
University of Michigan Ann Arbor Computing Centre. 1976.

Stephens, P.D.
The IMP Language and Compiler
Computer Journal vol. 17 no. 3. 1974.

Stevens, B.A.
A Note on Figure of Merit
Sigmetrics PER vol. 4 no. 1. 1975.

Tesdata 1976
Load Generator System Users' Manual
Tesdata System Corp. McLean, Virginia. 1976.

Turner, R.
Functional Specification for Script-II
Digital Equipment Corporation internal report. 1976.

Turner, R.
Test Definition Language
Digital Equipment Corporation internal report. 1976.

Turner, R. and Kosco, D.
Some Observations from a Throughput Experiment on the 11/70
Digital Equipment Corporation internal report 130-171-012-00. Jan. 1976.

Turner, R. and Levy, H.
Performance Evaluation of IAS on the PDP-11/70
Proc. International Symposium on Computer Performance, Modelling, Measurement and Evaluation, Harvard. 1976.

University of Michigan Ann Arbor Computing Centre 1976
Specification for a Terminal Simulator DSR

Vyssotsky, V.A., Corbato, F.J. and Graham, R.M.
Structure of the MULTICS Supervisor
AFIPS, FJCC vol. 27. 1965.

Waite, W.M.
A Sampling Monitor for Applications Programs
Software Practice and Experience vol. 3. 1973.

Watkins, S.W. and Abrams M.D.
Survey of Remote Terminal Emulators
US-NBS special publication 500-4. 1977.

Whitfield, H.
Conversations with the author. 1972.

Whitfield, H. and Wight, A.S.
EMAS - The Edinburgh Multi-Access System
Computer Journal vol. 16 no. 4. 1973.

Wight, A.S.
The EMAS Archiving Program
Computer Journal vol. 18 no. 2. 1975.

Wilkes, M.V.
The Cambridge Multiple Access System in Retrospect
Software Practice and Experience vol. 3 no. 4. 1973.

Wright, L. and Burnette, W.A.
An Approach to Evaluating Time Sharing Systems:
MH - TSS a Case Study
Sigmetrics PER vol. 5 no. 1. 1976.

Yates, F.
The Design and Analysis of Factorial Experiments
Commonwealth Bureau of Soils, technical communication
no. 35. 1937.